**FORSCHUNGSZENTRUM JÜLICH GmbH**
**Zentralinstitut für Angewandte Mathematik**
**D-52425 Jülich, Tel. (02461) 61-6402**

Interner Bericht

# Linux-Zugang mit Benutzer-Authentisierung

*Andrea Trensch*

FZJ-ZAM-IB-2004-10

September 2004

(letzte Änderung: 30.09.2004)

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

Der Zugang zu einem Computer oder der Zugriff auf geschützte Daten erfordern eine Anmeldung, mit der sich ein Benutzer gegenüber dem System identifiziert. Das übliche Login-Verfahren mit der Eingabe der UserID und eines Passwortes (*One Factor Authentication System*) wird heutigen Sicherheitsanforderungen nicht mehr gerecht. Das liegt sowohl an den Benutzern, die dazu neigen, einfache Passwörter zu wählen bzw. sich die sicheren, komplizierteren Passwörter aufzuschreiben, als auch an den steigenden Fähigkeiten der Angreifer, Passwörter zu knacken.
Mehr Sicherheit erlangt man, wenn der Benutzer neben dem Passwort eine Art Schlüssel besitzen muss, um sich zu authentisieren (*Two Factor Authentication System* - Wissen und Besitz). Der Vorteil solcher Verfahren liegt darin, dass ein Angreifer beide Komponenten benötigt, um sich unter diesem Account in das System einzuloggen.

In der vorliegenden Arbeit wird für den Einsatz eines *Two Factor Authentication System* im Bereich der Linux-Workstation-Gruppen im Forschungszentrum Jülich ein Konzept entwickelt und in Teilen implementiert. Vor- und Nachteile des Verfahrens werden diskutiert.
Dazu werden zuvor die Grundlagen zu kryptogaphischen Verfahren vorgestellt, die Benutzerverwaltung im Forschungszentrum Jülich für die Linux-Gruppen beschrieben und Sicherheitsmechanismen und Datenträger für die Speicherung von Schlüsseln untersucht.

# Kapitel 2

# Begriffe und Definitionen

## 2.1 Identifikation und Authentisierung

Damit ein Benutzer sich in ein System einwählen kann, muss zunächst festgestellt werden, ob er über die notwendigen Rechte verfügt. Dazu dient seine Identifikation und vor allem seine Authentisierung.

**Identifikation** Die Identifikation dient der eindeutigen Bestimmung der Identität eines Benutzers. Dies geschieht in der Regel durch Eingabe eines eindeutigen Login-Namens (UserID).

**Authentisierung** Die Authentisierung ist das ”Beweisen” der Identität eines Benutzers. Es soll festgestellt werden, ob die Person wirklich zum angegebenen Login-Namen gehört.

Es gibt drei Arten der Authentisierung:

- Geheimes Wissen (wie z.B. Passwörter oder Passphrases),
- Besitz (von Gegenständen, wie Chipkarten, Schlüssel) und
- personenbezogene biometrische Merkmale[1] (Fingerabdruck, Netzhaut, Gesicht)

Der dritte Ansatz ist noch Gegenstand der Diskussion, zum Beispiel bei der Wahl sicherer Verfahren für den Personalausweis. Er besitzt prinzipielle Vorteile hinsichtlich Verlust und Fälschungssicherheit, daher wird seine Bedeutung zunehmen.

Für maschinelle Authentisierung an der Mensch-Maschine-Schnittstelle, an der ein informationstechnisches System durch einen menschlichen Benutzer oder ein anderes informationstechnisches System zur Durchführung einer sicherheitsrelevanten Aufgabe aufgefordert wird, sind nach wie vor die Passwortprüfung und/oder der maschinenlesbare Ausweis verbreitete Methoden.

## 2.2 Autorisierung

Da Autorisierung sprachlich sehr leicht mit Authentisierung verwechselt wird, soll auch dieser Begriff hier erläutert werden.

[1] Biometrie - Lehre von der Zählung und (Körper-)Messung an Lebewesen [Duden]

Wenn ein Benutzer eines Systems identifiziert und authentisiert ist, regelt die Autorisierung, welchen Dienst er auf diesem System nutzen darf. Diese Information ist bei der Einrichtung des Benutzers auf einem System vom Administrator (im Auftrag) festgelegt worden.

## 2.3 Passwortwahl

Passwort-gestützte Verfahren zählen zu den meist verbreiteten Authentisierungsmechanismen und sind bei richtiger Passwortwahl in vielen Bereichen ausreichend. Bei einem 8-stelligen Passwort ergeben sich (bei z.B. 80 möglichen Tastatur-Zeichen) $80^8$ Kombinationen. Die Chance, ein solches durch systematisches Durchprobieren aller Tastenkombinationen (Brute-Force-Angriff) zu finden, ist verschwindend gering, da der Zeitaufwand exponentiell mit der Länge des Passwortes steigt. Geht man z.B. davon aus, dass ein System in einer Sekunde 1 Million Kombinationen ausprobieren könnte, so würde es durchschnittlich 25 Jahre dauern (maximal ca. 53 Jahre), das richtige Passwort zu finden. Bei einem nur 4 stelligen Passwort ergeben sich unter gleichen Bedingungen $80^4$ Kombinationen und ein maximaler Zeitaufwand von nur ca. 40 Sekunden.
Das zeigt, dass ein hinreichend langes, aus zufälligen Zeichen (Buchstaben, Zahlen und Sonderzeichen) bestehendes Passwort ausreichend sicher wäre.

Leider zeigt die Erfahrung, dass viele Benutzer trotz aller Warnungen und Hinweise nicht bereit sind, sich solche sicheren Passwörter auszudenken und zu merken. Stattdessen werden sogenannte schwache Passwörter, wie Namen von Freunden oder Haustieren verwendet, oder die sicheren Passwörter werden leicht auffindbar und ungeschützt aufgeschrieben.

Daher ist es sicherer, auf ein *Two Factor Authentication System* zurückzugreifen, wie es im Konzept in Kapitel 7 vorgestellt wird. Das Wissen des Passwortes allein nützt einem Angreifer nichts, er muss auch in den Besitz des ”Schlüssels” gelangen. In so einem Fall muss der Benutzer den Verlust des ”Schlüssels” rechtzeitig melden.

# Kapitel 3

# Kryptographie

Dieses Kapitel ist eine Einführung in die Kryptographie und in Verschlüsselungsverfahren, insbesondere in das RSA-Verfahrens, da das entwickelte Konzept (Kapitel 7) darauf aufbaut.

Die Kryptographie befasst sich mit dem Ver- und Entschlüsseln von Daten, um die enthaltenen Informationen für Unbefugte geheim zu halten und nur dem berechtigten Empfänger Zugriff auf den Klartext[1] zu erlauben. Zusätzlich dienen kryptographische Verfahren dazu, sicherzustellen, dass die übermittelte Nachricht nicht verändert wurde (Wahrung der Datenintegrität) und der Absender eindeutig identifiziert wurde (Feststellung der Authentizität).
Die Grundanforderungen an ein kryptographisches Verfahren sind, dass der durch die Verschlüsselung entstandene Chiffre-Text[2] wieder eindeutig in den Klartext zurückverwandelt werden kann und der Ver- und Entschlüsselungsvorgang mit möglichst wenig Aufwand und in vertretbarer Zeit durchführbar ist. Außerdem darf ein Unberechtigter aus dem Chiffretext keine Informationen erhalten, die den Entschlüsselungsmechanismus offenlegen könnten.

## 3.1 Symmetrische Verschlüsselung

Symmetrische Verschlüsselungsverfahren verwenden eine mathematische Vorschrift, um aus einem Klartext ($T_K$) einen chiffrierten Text ($T_C$) zu erzeugen.

$$T_C = ENCRYPT(Key, T_K)$$

Mit Hilfe eines Schlüssels ($Key$) wird jedes Zeichen oder auch jede Zeichengruppe umgewandelt. Der Empfänger benötigt denselben Schlüssel ($Key$) und kehrt diese Operation um:

$$T_K = DECRYPT(Key, T_C)$$

Für symmetrische Verfahren gibt es zwei Methoden:

- **Transposition**: Die Reihenfolge der Klartextzeichen wird verändert.

  Zum Beispiel wird der Text zeilenweise in eine Matrix geschrieben und spaltenweise wieder ausgelesen (Spaltentransposition):

```
INFORMATIK     =>     INFOR
                      MATIK     =>     IMNAFTOIRK
```

[1] Ursprüngliche Mitteilung vor der Verschlüsselung
[2] Die Mitteilung (der Klartext) nach der Verschlüsselung

Bei der Transposition sind somit alle Zeichen des Ciffretextes auch Element des Klartextes (bis auf eventuelle Blindzeichen, die zum Auffüllen der Matrix verwendet werden können). Der *Schlüssel* ist die Anzahl der Spalten der Matrix.

- **Substitution**: Die Zeichen an sich werden auf ihrer Position verändert.

  Das heißt, es wird ein Geheimtextalphabet benötigt, zum Beispiel die Verschiebung aller Buchstaben um 3 Stellen nach rechts (*Schlüssel*information):

  ```
  Klartextalphabet:

      ABCDEFGHIJKLMNOPQRSTUVWXYZ      =>      INFORMATIK

  Geheimtextalphabet:

      YZABCDEFGHIJKLMNOPQRSTUVWX      =>      GLDMPKYRGI
  ```

  Bei der Substitution behalten die Buchstaben die Position des Klartextes bei, auch wenn sie durch andere ersetzt werden. Die sprachlichen Eigenschaften und die Häufigkeit des Auftretens bestimmter Buchstaben werden nicht verändert.

Ein Nachteil dieser Verfahren ist unter anderem die Verteilung des Schlüssels an den berechtigten Empfänger, da kein Unberechtigter den Schlüssel kennen darf. Dies erfordert einen geheimen und sicheren Transport des Schlüssels.
Außerdem besteht die Gefahr des Missbrauchs: Ein Dritter, der diesen Schlüssel kennt, kann Nachrichten des Absenders vortäuschen.

Diese Probleme umgehen asymmetrische Verfahren.

## 3.2 Asymmetrische Verschlüsselung

Asymmetrische Verschlüsselungsverfahren, auch **Public-Key-Verfahren** genannt, verwenden nicht nur einen Schlüssel, sondern ein Schlüsselpaar. Jeder Teilnehmer besitzt ein solches Paar, das aus einem geheimen (private Key) und einem öffentlichen (public Key) Schlüssel besteht. Letzterer darf für alle frei sichtbar sein, daher ist auch der Transport über unsichere Datennetze kein Problem.

Die Verschlüsselung des Klartextes $T_K$ erfolgt mit dem öffentlichen Schlüssel $PubKey$ des Empfängers

$$T_C = ENCRYPT(PubKey, T_K)$$

und kann nur mit dem zugehörigen geheimen Schlüssel $PrivKey$ vom berechtigten Empfänger zurückverwandelt werden.

$$T_K = DECRYPT(PrivKey, T_C)$$

Selbst der Absender kann nach dem Verschlüsseln seine eigene Nachricht nicht wieder entschlüsseln.

## 3.3 Vergleich

Jeder Teilnehmer eines asymmetrischen Verschlüsselungsverfahrens muss nur einen einzigen Schlüssel sicher aufbewahren, seinen privaten. Die öffentlichen Schlüssel aller Teilnehmer können in einem gemeinsamen Verzeichnis für alle zugänglich abgelegt werden.
Bei den symmetrischen Verfahren dagegen benötigt man eine Vielzahl von Schlüsseln, wenn man

mit jedem einzelnen Teilnehmer einer größeren Gruppe geheim kommunizieren möchte.

Ein weiterer Vorteil der Public-Key-Verfahren ist, dass Daten und Nachrichten nicht nur geheim gehalten werden können, sondern dass in Form der Signatur auch die Datenintegrität (Echtheit der Daten, d.h. sie wurden nicht verändert) und die Integrität des Absenders (kein Vortäuschen fremder Identität) gewahrt bleiben.

Die asymmetrischen Verfahren gelten heutzutage als sicher. Sie benutzen zwei sehr große Zahlen, die multipliziert oder potenziert werden. Die Umkehrung dieses einfachen Schrittes gilt als unmöglich. Das heißt, das Produkt zweier sehr großer Zahlen, ohne Kenntnisse dieser, wieder zu zerlegen (zu faktorisieren) bzw. den Logarithmus einer Zahl zu ermitteln, die aus dem Potenzieren zweier großer Zahlen entstanden ist, kann in begrenzter Zeit nicht durchgeführt werden.

Im folgenden Abschnitt wird ein Vertreter der asymmetrischen Verfahren genauer erklärt.

## 3.4 RSA-Verfahren

In der Umsetzung des entwickelten Konzeptes für ein *Two Factor Authentication System* in Kapitel 8 werden Funktionen verwendet, die den RSA-Algorithmus implementiert haben. Daher wird an dieser Stelle genauer auf dieses Verfahren eingegangen.

Der RSA-Algorithmus wurde 1978 von seinen Namensgebern Ron Rivest, Adi Shamir und Leonard Adleman veröffentlicht und ist wohl das bekannteste und am weitesten verbreitetste asymmetrische Verschlüsselungsverfahren [Lit. 4]. Es verwendet zwei große Primzahlen und seine Sicherheit basiert auf der Schwierigkeit, deren Produkt zu faktorisieren.

### 3.4.1 Grundlage

Der RSA-Algorithmus basiert auf folgender Tatsache [Lit. 5]:

*Sei n = pq das Produkt zweier verschiedener Primzahlen p und q. Dann gilt für jede natürliche Zahl* $m \leq n$ *und jede natürliche Zahl k die Gleichung*

$$m^{k(p-1)(q-1)+1} \textit{ mod } n = m$$

### 3.4.2 Schlüsselerzeugung

Um ein Schlüsselpaar zu erzeugen, müssen folgende Schritte ausgeführt werden:

1. Es werden zwei große Primzahlen $p$ und $q$ gewählt.

   Die Primzahlen sollten mindestens 1024 Bit groß sein, also 100 oder mehr Dezimalziffern haben.

   Das Erstellen der Primzahlen wird normalerweise mit Hilfe von Zufallszahlen bewerkstelligt. Dabei werden zufällig zwei Zahlen der erforderlichen Größe ausgewählt und es wird getestet, ob es sich um Primzahlen handelt. Da ein vollständiges Überprüfen der Zahlen aber

zu lange dauern würde, greift man auf statistische Tests zurück. Mit ihrer Hilfe kann die Wahrscheinlichkeit festgestellt werden, mit der es sich um eine Primzahl handelt. Einer der bekanntesten ist der Rabin-Miller Test. Besteht eine Zahl diese Überprüfung, so ist sie zu 99,9% eine Primzahl, ansonsten mit Sicherheit nicht.

2. Es wird das Produkt $n = p * q$ gebildet.

   $n$ ist N Bit lang und wird als Modulus bezeichnet.

3. Dann wird ein $e$ mit $1 < e < n$ so gewählt, dass es zu $(p-1)(q-1)$ teilerfremd ist.

   Daraus ergibt sich, dass $e$ ungerade ist, denn das Produkt $(p-1)(q-1)$ ist zwangsläufig gerade. $e$ muss keine Primzahl sein, läßt sich aber z.B. über eine Primfaktorzerlegung des Produktes $(p-1)(q-1)$ berechnen, mit der alle Primfaktoren ermittelt werden und so eine teilerfremde Primzahl als $e$ ausgewählt werden kann.

4. Berechnen eines $d$ mit $d * e = 1 \bmod ((p-1) * (q-1))$.

   Das heißt, $d * e - 1$ muss ohne Rest durch $(p-1) * (q-1)$ teilbar sein.
   Außerdem soll $d \neq e$ sein.
   Um $d$ zu berechnen, muss eine ganze Zahl $x$ gefunden werden, so dass die folgende Gleichung ein ganzzahliges Ergebnis liefert:
   $d = (x * (p-1) * (q-1) + 1)/e$

*Die Zahlen **n** und **e** bilden den öffentlichen Schlüssel (**PubKey**),*
*die Zahlen **n** und **d** den privaten Schlüssel (**PrivKey**).*

Dies erklärt auch die Zusatzforderung, dass $e$ und $d$ nicht den gleichen Wert haben sollen, denn sonst wären öffentlicher und privater Schlüssel gleich.

Nach der Berechnung der Schlüssel müssen alle Hilfsgrößen wie $p$, $q$, $(p-1) * (q-1)$ und $x$ gelöscht werden.

### 3.4.3 Datenverschlüsselung

Das Verschlüsseln erfolgt mit Hilfe des *öffentlichen Schlüssels*, bestehend aus dem öffentlichen Exponent $e$ und dem Modulus $n$. Der Ablauf sieht wie folgt aus:

1. Unterteilung des Klartextes $T_K$ in Blöcke $k$, die kleiner der Länge N Bit sind (Schlüssellänge sei N Bit).

   Buchstaben und sonstige Zeichen müssen vorher mittels einer Tabelle in Zahlencodes umgewandelt werden (siehe Beispiel Seite 10).

2. Verschlüsselung jeden dieser Blöcke $k$ durch Anwenden der Formel
   $c = k^e \bmod n$.

   Die Formel besagt, dass der Klartextblock $k$ mit $e$ zu potenzieren und dann durch $n$ zu teilen ist und der Rest den chiffrierten Block $c$ ergibt.

3. Zusammensetzen der chiffrierten Blöcke $c$ zu einer Nachricht.

Zu beachten ist, dass die Anzahl der Bits der verschlüsselten Blöcke größer ist, als die der Klartextblöcke.

### 3.4.4 Entschlüsselung

Um einen chiffrierten Text $T_C$ wieder zu entschlüsseln, wird der *private Key*, bestehend aus dem privaten Exponent $d$ und dem Modulus $n$, benötigt. Das Entschlüsseln ist nahezu identisch mit der Verschlüsselung:

1. Unterteilung des Ciffretextes in Blöcke $c$ von N Bit Länge (Schlüssel hat die Länge N Bit).
2. Entschlüsselung jeden dieser Blöcke $c$ mit Hilfe folgender Formel:
   $k = c^d \bmod n$.
3. Zusammensetzen der dechiffrierten Blöcke $k$ zu der Ausgangs-Nachricht.

Beim Zusammensetzen der entschlüsselten Klartextblöcke muss berücksichtigt werden, dass diese jeweils aus weniger Bits bestehen.

### 3.4.5 Beweis

Die Korrektheit des Entschlüsselns als Umkehrung zum Verschlüsseln, läßt sich mit Hilfe des Satzes von Euler nachvollziehen [Lit. 5]:

| | |
|---|---|
| Verschlüsselung: | $f_e(m) := m^e \bmod n$ |
| Entschlüsselung: | $f_d(m) := c^d \bmod n$ |
| | |
| Korrektheit: | $f_d(f_e(m)) = (m^e)^d \bmod n = m$ |

### 3.4.6 Signatur

Wie am Anfang dieses Abschnittes erwähnt, dienen kryptographische Verfahren nicht nur zur Geheimhaltung von Informationen vor Unbefugten, sondern mit ihnen kann man auch die Echtheit einer Nachricht durch Feststellung der Authentizität des Absenders überprüfen. Dies geschieht, indem der Absender die Nachricht signiert. Die Signatur erfolgt mit dem *privaten* Schlüssel des Absenders (privater Exponent $d$ und Modulus $n$) und kann somit nur von ihm durchgeführt werden, wenn er den privaten Schlüssel stets geheim hält.
Die Entschlüsselung der Signatur nennt man **Verifikation** und jeder kann sie mit dem *öffentlichen Schlüssel* des Absenders (öffentlicher Exponent $e$ und Modulus $n$) durchführen.

Der Ablauf ist analog zur Datenverschlüsselung:

Die Nachricht wird in Blöcke $k$ unterteilt und ergibt die chiffrierten Blöcke $c$.

Die Formeln sehen dann wie folgt aus:

- Verschlüsselung (Signatur):
  $c = k^d \bmod n$
- Entschlüsselung (Verifikation):
  $k = c^e \bmod n$

### 3.4.7 Vor- und Nachteile des RSA-Verfahrens

Die Sicherheit des RSA-Verfahrens besteht nicht in der Geheimhaltung des Algorithmus, sondern in der Schwierigkeit für Angreifer, in endlicher Zeit den Schlüssel zu berechnen.

Der Zeitaufwand für das Ver- und Entschlüseln ist gegenüber anderen Verschlüsselungsverfahren höher (zum Beispiel um Faktor 100 bis 1000 langsamer als **DES**[3] [Lit. 6]). Bei den heutigen Rechenleistungen kann man diese geringere Geschwindigkeit des RSA-Verfahrens allerdings nicht mehr als Problem ansehen.

### 3.4.8 Beispiel

An dem folgenden Beispiel soll nur das Prinzip des RSA-Verfahrens verdeutlicht werden, deshalb werden einfache Zahlen verwendet, die für die Realität nicht ausreichend sind.

#### Schlüsselerzeugung

(1.) Für die beiden Primzahlen werden
$p = 11$ und $q = 5$ gewählt.

(2.) Daraus ergibt sich $n$:
$n = p * q = 55$

(3.) $(p-1) * (q-1) = 10 * 4 = 40$
Für $e$ wird die Primzahl 7 gewählt, da die Primzahlzerlegung
von $40 = 2^3 * 5$ ergibt.
Somit ist $e$ teilerfremd zu $(p-1) * (q-1)$.

(4.) Für ein $d = 23$ ist die Bedingung erfüllt:
$d * e \ \ mod \ \ ((p-1) * (q-1)) \quad = \quad 161 \ \ mod \ \ 40 \quad = \quad 1$

Es wurden folgende Größen berechnet:

| | |
|---|---|
| p | 11 |
| q | 5 |
| n | 55 |
| e | 7 |
| d | 23 |

$n$ und $e$ bilden den öffentlichen Schlüssel,
$n$ und $d$ bilden den privaten Schlüssel.

**Tabelle 3.1:** *Auflistung aller berechneten Schlüsselgrößen*

#### Verschlüsselung

Als Beispiel wird das Wort

I N F O R M A T I K

verschlüsselt.

[3] Data Encryption Standard - Ein von IBM entwickeltes symmetrisches Chiffrierverfahren, das 1976 offiziell als US-Standard übernommen wurde

Die Buchstaben müssen vor der Verschlüsselung in Zahlen umgewandelt werden. Für das Beispiel gilt folgende Tabelle.

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 |
| P | Q | R | S | T | U | V | W | X | Y | Z | Ä | Ö | Ü | |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | |
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 |
| p | q | r | s | t | u | v | w | x | y | z | | | | |
| 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | | | | |

**Tabelle 3.2:** *Zuordnung der Buchstaben zu Zahlenblöcken für das Beispiel*

Daraus ergibt sich die Zuordnung

```
 I  N  F  O  R  M  A  T  I  K
09 14 06 15 18 13 01 20 09 11
```

Die einzelnen zu verschlüsselnden Elemente müssen in Stücke zerlegt werden, die kleiner als $n$ sind, also kleiner als $55$. Das heißt, es werden in diesem Beispiel die zu jedem Buchstabenblock gehörenden Zahlen beibehalten. Wenn man ein größeres $n$ als Modulus wählt, zum Beispiel eine vierstellige Zahl, dann können jeweils zwei Buchstaben zu einem Block zusammengefasst werden, bevor sie verschlüsselt werden.

Nach Anwendung der Formel

$$c = k^e \bmod n$$

ergeben sich Chiffreblöcke (Rechnung siehe Maple-Output in Abb. 3.1 am Ende des Kapitels), die anhand der Tabelle 3.2 wieder in Buchstaben umgewandelt werden können.

| | | |
|---|---|---|
| $09^7 = 0004782969$ | $09^7 \mod 55 = 04$ | D |
| $14^7 = 0105413504$ | $14^7 \mod 55 = 09$ | I |
| $06^7 = 0000279936$ | $06^7 \mod 55 = 41$ | l |
| $15^7 = 0170859375$ | $15^7 \mod 55 = 05$ | E |
| $18^7 = 0612220032$ | $18^7 \mod 55 = 17$ | Q |
| $13^7 = 0062748517$ | $13^7 \mod 55 = 07$ | G |
| $01^7 = 0000000001$ | $01^7 \mod 55 = 01$ | A |
| $20^7 = 1280000000$ | $20^7 \mod 55 = 15$ | O |
| $09^7 = 0004782969$ | $09^7 \mod 55 = 04$ | D |
| $11^7 = 0019487171$ | $11^7 \mod 55 = 11$ | K |

**Tabelle 3.3:** *Verschlüsselung der Klartext-Blöcke*

Somit sieht das zu verschlüsselnde Wort

I N F O R M A T I K

als Chiffretext so aus:

D I l E Q G A O D K

**Entschlüsselung**

Zur Kontrolle wird das entstandene Wort nach folgender Vorschrift wieder dechiffriert.

$$k = c^d \bmod n$$

Da bereits mit diesen für die Praxis unrealistisch kleinen Zahlen als Schlüssel ($n$, $e$ und $d$) Zwischenergebnisse mit über 30 Ziffern entstehen, ist auch die Rechnung der Entschlüsselung in der Abbildung 3.1 aufgeführt.

Die Ergebnisse werden in der folgenden Tabelle zusammengefasst:

| | |
|---|---|
| $04^{23} \bmod 55 = 09$ | I |
| $09^{23} \bmod 55 = 14$ | N |
| $41^{23} \bmod 55 = 06$ | F |
| $05^{23} \bmod 55 = 15$ | O |
| $17^{23} \bmod 55 = 18$ | R |
| $07^{23} \bmod 55 = 13$ | M |
| $01^{23} \bmod 55 = 01$ | A |
| $15^{23} \bmod 55 = 20$ | T |
| $04^{23} \bmod 55 = 09$ | I |
| $11^{23} \bmod 55 = 11$ | K |

**Tabelle 3.4:** *Entschlüsselung der Chriffe-Blöcke*

## 3.5 Zertifikate

Wie unter 3.4.6 beschrieben, bieten asymmetrische Verfahren - wie das RSA-Verfahren - die Möglichkeit, durch das Signieren die Authentizität des Absenders zu beweisen. Diese digitale Signatur ist Teil der digitalen Identität [Lit. 6]. Um diese Identität beweisen zu können, benötigt man einen elektronischen Ausweis, in dem - vergleichbar mit dem Personalausweis - alle wichtigen personenbezogenen Daten gespeichert werden. Dieser elektronische Ausweis wird **Zertifikat** genannt. Damit wird neben dem Namen, der E-Mail-Adresse und anderen Angaben auch die digitale Signatur bestätigt.

Die Bestätigung führt eine Zertifizierungsstelle durch (CA - Certificate Authority), indem sie das Zertifikat signiert und diesem dann vertraut werden kann. Als oberste Instanz einer hierarischen Vertrauenskette gibt es die Root CA, deren Zertifikat vertraut werden muss [Lit. 7]. Alle weiteren Zertifikate bauen auf diesem auf.

Im praktischen Gebrauch haben sich in der letzten Zeit die so genannten **X.509-Zertifikate** durchgesetzt. Sie enthalten im Allgemeinen folgende Angaben [Lit. 6]:

- Name des Zertifizierungs-Objektes
- Name der Zertifizierungsstelle
- Öffentlicher Schlüssel des Zertifizierungs-Objektes
- Digitale Unterschrift der ausgebenden Stelle
- Gültigkeitsdauer des Zertifikats
- Seriennummer des Zertifikats

Es können weitere Informationen hinzugefügt werden, um die Zertifikate für die verschiedenen Anwendungen brauchbar zu machen.

Für einen Login-Vorgang reichen die Informationen eines X.509-Zertifikates nicht aus, da wichtige Daten für die Autorisierung , wie UserID, erlaubte Hosts, etc. anderweitig verwaltet werden müssen (siehe Kapitel 4 ”Benutzerverwaltung”).

```
> restart:

Codierung des Wortes  I N F O R M A T I K  mit Zahlen
-------------------------------------------------------------
> text := [09, 14, 06, 15, 18, 13, 01, 20, 09, 11];
                                  text := [9, 14, 6, 15, 18, 13, 1, 20, 9, 11]
Schlüsselgrößen
----------------------
> n := 55: e := 7: d := 23:
  printf("\nn = %d\ne = %2d\nd = %d\n\n",n,e,d);

n = 55
e =  7
d = 23

Verschlüsselung
----------------------
> printf("\n      text ^ e     text ^ e mod n
  \n--------------------------------");
  for i from 1 to nops(text) do
    enc[i] := text[i] ^ e:
    crypt[i] := enc[i] mod n:
    printf("%14d          %2d \n",enc[i], crypt[i]);
   end do:

      text ^ e     text ^ e mod n
--------------------------------
       4782969           4
     105413504           9
        279936          41
     170859375           5
     612220032          17
      62748517           7
             1           1
    1280000000          15
       4782969           4
      19487171          11

Entschlüsselung
----------------------
> printf("\n                                        crypt ^ d     crypt ^ d mod n
  \n-------------------------------------------------------------------");
  for i from 1 to nops(text) do
    dec[i] := crypt[i] ^ d;
    k_text:= dec[i] mod n;
    printf ("%40d          %2d \n",dec[i],k_text);
  end do:

                                crypt ^ d     crypt ^ d mod n
-------------------------------------------------------------
                          70368744177664           9
                 8862938119652501095929          14
  1241734376913948688227832002063214972l           6
                    11920928955078125          15
           19967568900859523802559065713          18
                    27368747340080916343          13
                                       1           1
            1122274146401882171630859375          20
                          70368744177664           9
                89543024325523737224653l          11
```

**Abbildung 3.1:** *Maple-Programm zur Berechnung der Chiffre-Blöcke bei der Verschlüsselung und der Text-Blöcke bei der Entschlüsselung*

# Kapitel 4

# Benutzerverwaltung

Dieses Kapitel gibt einen Überblick über die Benutzerverwaltung und das Konzept der Workstation-Gruppen, da in dieser Umgebung das erweiterte Login-Verfahren implementiert werden soll.

In vielen Rechenzentren ist die Benutzerverwaltung zentral organisiert. Im Forschungszentrum Jülich (FZJ) übernimmt diese Aufgabe das Dispatch im Zentralinstitut für Angewandte Mathematik (ZAM). Es ist unter anderem verantwortlich für die Einrichtung von Benutzerkennungen und die Vergabe von Rechten und Betriebsmitteln.
Die Linux-Rechner an den Arbeitsplätzen der Benutzer werden im Rahmen von Workstation-Gruppen (WSG) betreut.

## 4.1 Workstation-Gruppen

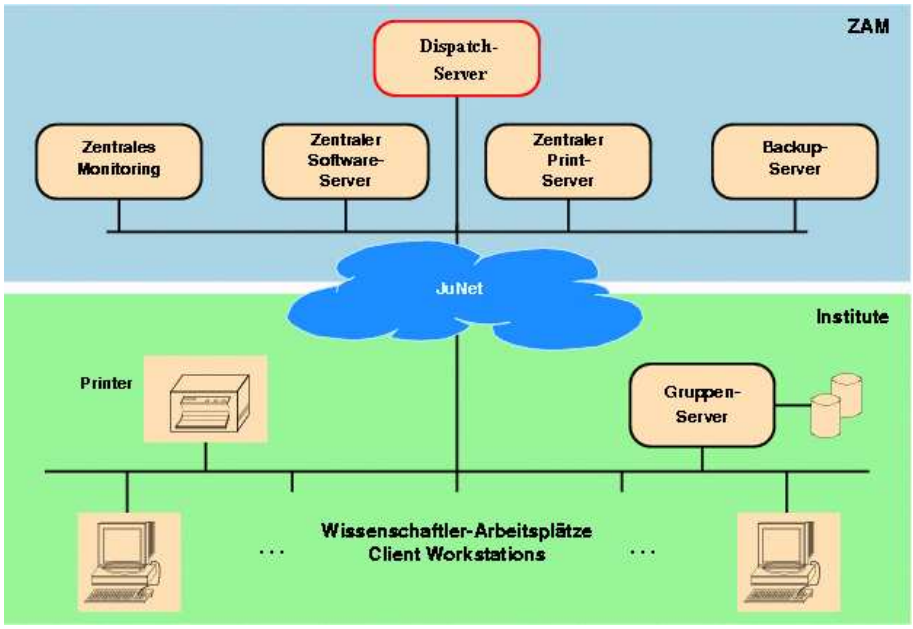
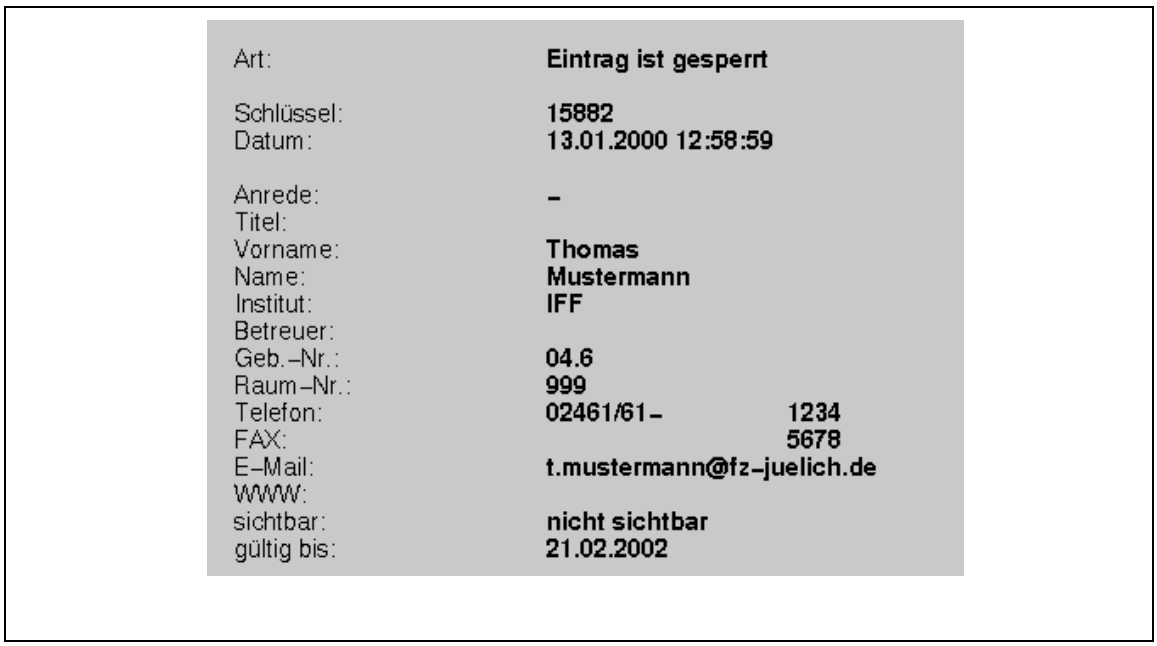



**Abbildung 4.1:** *Konzept der Workstation-Gruppe*

In einer Workstation-Gruppe (Abbildung 4.1) sind alle zugehörigen Arbeitsplatzrechner mit einem Gruppenserver verbunden. Die Gruppen werden zentral über das JuNet (Datennetz des FZJ) vom ZAM betreut, das über zentrale Server Daten und Software an die Gruppenserver exportiert, die die Benutzerdaten und Software an die Arbeitsplatzrechner weiter geben.

Der Gruppenserver ist ein NIS[1]/NFS[2]-Server, der den Zugang zu den Arbeitsplätzen regelt. Das heißt, wenn ein Benutzer auf dem NIS-Server bekannt ist, kann er sich mit nur einer Kombination von UserID und Passwort auf allen Rechnern in der Gruppe anmelden.

## 4.2 Benutzerdatenbank des Rechenzentrums



| | |
|---|---|
| Art: | **Eintrag ist gesperrt** |
| Schlüssel: | **15882** |
| Datum: | **13.01.2000 12:58:59** |
| Anrede: | **–** |
| Titel: | |
| Vorname: | **Thomas** |
| Name: | **Mustermann** |
| Institut: | **IFF** |
| Betreuer: | |
| Geb.–Nr.: | **04.6** |
| Raum–Nr.: | **999** |
| Telefon: | **02461/61– 1234** |
| FAX: | **5678** |
| E–Mail: | **t.mustermann@fz–juelich.de** |
| WWW: | |
| sichtbar: | **nicht sichtbar** |
| gültig bis: | **21.02.2002** |

**Abbildung 4.2:** *User-Daten Eintrag*

In der Benutzerdatenbank gibt es für jeden Nutzer einen Datenbank-Record, der typischerweise folgende Einträge enthält (Ausschnitt in der Abbildung 4.2):

- Name
- E-Mail-Adresse
- Institut
- Telefonnummer
- Gültigkeit
- Login-Accounts für Ressourcen u.a.

Die Benutzer werden über das Dispatch-System in arbeitsgruppen-spezifische NIS-Tabellen eingetragen, die über das Netzwerk auf die Gruppen-Server der jeweiligen Workstationgruppe verteilt werden.

## 4.3 Die Dispatch-Struktur

Neben den Benutzer-Daten stehen dem Dispatch Verwaltungstools, Web-basierte Formulare für die Anmeldung, sowie Prozeduren zur Einrichtung und Bearbeitung von User-Accounts zur Verfügung (Abbildung 4.3).

[1] NIS - Network Information Service
[2] NFS - Network File System

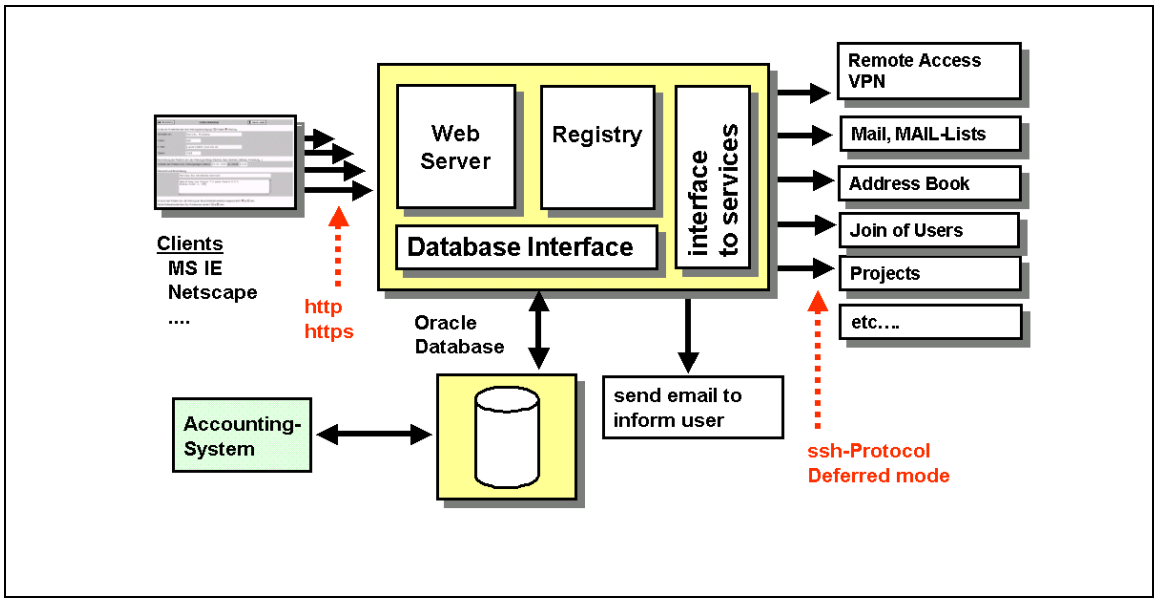



**Abbildung 4.3:** *Struktur des zentralen Dispatch-Systems*

Die Benutzer beantragen über Online-Formulare (im Bild oben links) die Dienste des Dispatch. Dazu gehören zum Beispiel die Beantragung der offiziellen E-Mail-Adresse und eines Rechner-Accounts, Ressourcenvergabe für zentrale Rechnersysteme, Eintragen oder Ändern der Adressdaten, usw. Die Anträge werden über das Web an das Dispatch geschickt. Dieses registriert die Benutzerangaben und trägt sie in die Datenbank ein (unten mitte). Diese Einträge werden u.a. für das Accounting benötigt (unten links). Über automatierte ssh-Aufrufe (siehe 6.1.2) werden die entsprechenden Aufgaben erfüllt, zum Beispiel das Joinen der Benutzer.

# Kapitel 5

# Login-Verfahren bei Linux

In diesem Kapitel wird ein Überblick über Login-Verfahren bei Linux-Systemen gegeben, wie sie auch in den Linux-Workstation-Gruppen (siehe voriges Kapitel) im FZJ verwendet werden. Das im zweiten Teil dieses Kapitels beschriebene Verfahren mit Linux-PAM wird im Konzept des praktischen Teils der Arbeit für das Two Factor Authentication System verwendet.

## 5.1 Linux User-Management

Linux ist ein Multi-User-System, d.h. es können gleichzeitig mehrere Benutzer am System arbeiten. Deshalb muss ein Benutzer vom System eindeutig identifiziert werden können. Zu diesem Zweck meldet sich jeder Benutzer mit einem Benutzernamen und mit einem Passwort an [Lit. 1].
Der Systemadministrator hat die Möglichkeit, für den Benutzer Voreinstellungen zu treffen. Diese stehen in der Datei `/etc/login.defs`. Für das Passwort gibt es zum Beispiel folgende Voreinstellungen:
PASS_MAX_DAYS und PASS_MIN_DAYS für die maximale bzw. minimale Gültigkeitsdauer und PASS_MIN_LEN und PASS_MAX_LEN für die Minimal- und Maximallänge des Passwortes. Zu beachten ist, dass die Maximallänge grundsätzlich 8 Zeichen beträgt, auch wenn ein größerer Wert angegeben wird. Werden dennoch längere Passwörter gewünscht, muss ein anderer Verschlüsselungsmechanismus für das Passwort eingestellt werden, z.B. mittels PAM (siehe Abschnitt 5.2).

#### Die Dateien der Benutzerverwaltung bei Linux

Informationen über Benutzer und Gruppen werden in 4 Dateien gespeichert:
`/etc/passwd` `/etc/shadow` `/etc/group` `/etc/gshadow`

**/etc/passwd** In der Vergangenheit wurden die Linux-Benutzer in einer einzigen Datei verwaltet. Dort stand der Benutzername, UserID (UID), Home-Verzeichnis, Standard-Shell und das verschlüsselte Passwort.
Das Passwort wird mittels der Funktion crypt (siehe Beschreibung mit ”man 3 crypt”) verschlüsselt.
Das Hauptproblem dieser Datei in Bezug auf Sicherheit ist die Bedingung, dass sie für alle lesbar sein muss, damit ein Benutzer herausfinden kann, wem eine bestimmte Datei gehört. Dazu muss er die UID (nur die wird im Inode einer Datei gespeichert) einem Benutzernamen zuordnen können. Diese Zuordnung erfolgt mit der Datei `/etc/passwd`.
Obwohl in dieser Datei die Passwörter der Benutzer in verschlüsselter Form vorliegen, war diese Art der Speicherung immer wieder Angriffspunkt erfolgreicher Ein-

bruchsversuche. Zum Beispiel wurden Listen von häufig gebrauchten Passwörtern benutzt und diese Passwörter nach dem selben Algorithmus verschlüsselt, wie beim Linux-System und dann mit den Einträgen in der `/etc/passwd`-Datei verglichen.
Deshalb wurde das Passwortfeld in eine eigene Datei ausgelagert, die nur für `root` lesbar ist. Anstelle des Passwortfeldes steht an dieser Stelle in der `/etc/passwd` ein `x`, das darauf hinweist, dass das Passwort in der Datei `/etc/shadow` steht.

**/etc/shadow** Diese Datei enthält die verschlüsselten Passwörter der Benutzer und andere Informationen zum Passwort und sollte nur für den Benutzer `root` lesbar sein.

Da die Verwaltung der Benutzer über zwei Dateien erfolgt, müssen alle Benutzer auch in beiden Dateien geführt werden.

Für die Verwaltung von Gruppen gibt es analog die Dateien `/etc/group` und `/etc/gshadow`.

#### Das Kommando `passwd`

Mit dem Kommando `passwd` kann man das Passwort eines Benutzers ändern. Wird der Befehl ohne Benutzernamen als Argument aufgerufen, kann der jeweilige Benutzer sein eigenes Passwort ändern. Weiterhin kann man mit der Option `-l` (`lock`) einen Benutzer sperren und mit der Option `-u` (`unlock`) wieder aktivieren.

## 5.2 Zielsetzung von PAM

Der verbreitetste Login-Mechanismus ist die Eingabe des Login-Namens und eines Passwortes über die Tastatur. Bei einem Linux-System steuert meist das Programm `login` diesen Vorgang. Das eingegebene Passwort wird verschlüsselt und mit dem in `/etc/shadow` (siehe Abschnitt 5.1) hinterlegten verschlüsselten Passwort verglichen. Stimmen diese beiden überein, wird der Zugang zum System gewährt.

Es gibt zahlreiche Anwendungen, die eine Authentifizierung des Benutzers erfordern. Wenn jede Anwendung ihre eigene Kombination von Benutzernamen und Passwort besitzt und sich selbst um Identifizierung und Authentifizierung ihrer Nutzer kümmert, steigt der Programmieraufwand unnötig und Administrator und Benutzer verlieren schnell den Überblick.
Möchte man ein anderes Authentifizierungs-Verfahren einsetzen, zum Beispiel auf einen NIS-Server zurückgreifen oder eine Smartcard bzw. einen USB-Stick verwenden, müßte im Regelfall jedes Programm im Source-Code angepasst und neu kompiliert werden. Auch mögliche Fehler im Login-Verfahren sind in allen Programmen zu beheben.
Das ist gerade im Hinblick auf die sich ständig ändernden Sicherheitsanforderungen nicht sehr flexibel.
Um mit Linux einen flexiblen und zentralen Authentifizierungsmechanismus realisieren zu können, entstand das Linux-PAM-Projekt [Lit. 2].

#### Was ist PAM?

Linux-PAM (**Pluggable Authentication Modules for Linux**) ist eine Sammlung von Modulen, die in sogenannten "`shared libraries`" gespeichert sind, und dient zur Benutzer-Authentif-

ikation. PAM bildet eine Software-Schicht mit klar definierten Schnittstellen [Lit. 3] zwischen der Anwendung (z.B. `login`) und dem aktuell verwendeten Authentifizierungs-Mechanismus. Bei einem PAM-kompatiblen Programm werden die für die Authentifikation zuständigen Funktionen in eine modulare Bibliothek ausgelagert und es wird eine Konfigurationsdatei mit den benötigten Modulen erstellt.
Die Konfiguration der Module ist für alle Anwendungen (z.B. Konsolen- oder Fenster-Login, Telnet, FTP, SSH) einheitlich strukturiert.
Um diese Entkopplung der Anwendung von der tatsächlich verwendeten Authentifizierung zu erreichen, wird ein universelles Interface genutzt, das unabhängig von dem gewünschten Authentifizierungsmechanismus ist.

Zu jeder Anwendung gehört eine PAM-Konfigurationsdatei, in der durch die Reihenfolge der abzuarbeitenden Module gezielt festgelegt werden kann, wie für einzelne Anwendungen Benutzer authentifiziert und Passwörter geändert werden.
Da sich die Anwendung nicht mehr selbst um die Authentifizierung kümmern muss, lassen sich neue Verfahren einbinden, ohne die Anwendung zu ändern. Allerdings ist PAM rein passiv und muss immer von einer Anwendung aufgerufen werden.
Ein weiterer Vorteil gegenüber der klassischen Passwort-Variante ist, dass PAM nicht nur das Authentication-Management übernimmt, sondern sich auch um das Account-, Session- und Passwort-Management kümmert.

### 5.2.1 Kommunikationsfluss zwischen der Anwendung und PAM

Jedes Programm, z.B. Konsolen-Login, lädt bei Bedarf die PAM-Bibliothek (`/lib/libpam.so`), diese wiederum greift auf die PAM-Module (`/lib/security/*.so`) zurück, die in der zugehörigen Konfigurationsdatei festgelegt sind.
Das PAM-Modul kommuniziert mit der PAM-Bibliothek, um seine Parameter auszulesen, und mit der Anwendung, um an die Benutzerdaten wie Accountname und Passwort zu gelangen.
Manche Module greifen zudem auf vorhandene Dienste zu oder lesen Dateien wie `/etc/passwd`.
Der Ablauf ist in der Abbildung 5.1 dargestellt.

### 5.2.2 Konfiguration der Authentifizierungsmechanismen

Für jede PAM-fähige Anwendung gibt es unter `/etc/pam.d` eine Konfigurationsdatei, die den gewünschten Authentifizierungsmechanismus beschreibt. In dieser Datei steht, welche PAM-Module in welcher Reihenfolge verwendet werden und gegebenenfalls wie auf das Ergebnis reagiert werden soll.
Die eigentliche Authentifizierung übernimmt dann nicht mehr die Anwendung, sondern das PAM-Modul. Bei Änderungen muss jeweils nur die PAM-Konfigurationsdatei geändert werden, es entfällt das umständliche Neukompilieren und somit kann man sehr flexibel auf neue Sicherheitsanforderungen reagieren.
Gewöhnlich trägt die Konfigurationsdatei den gleichen Namen wie der Dienst bzw. die Anwendung, z.B. "`login`".

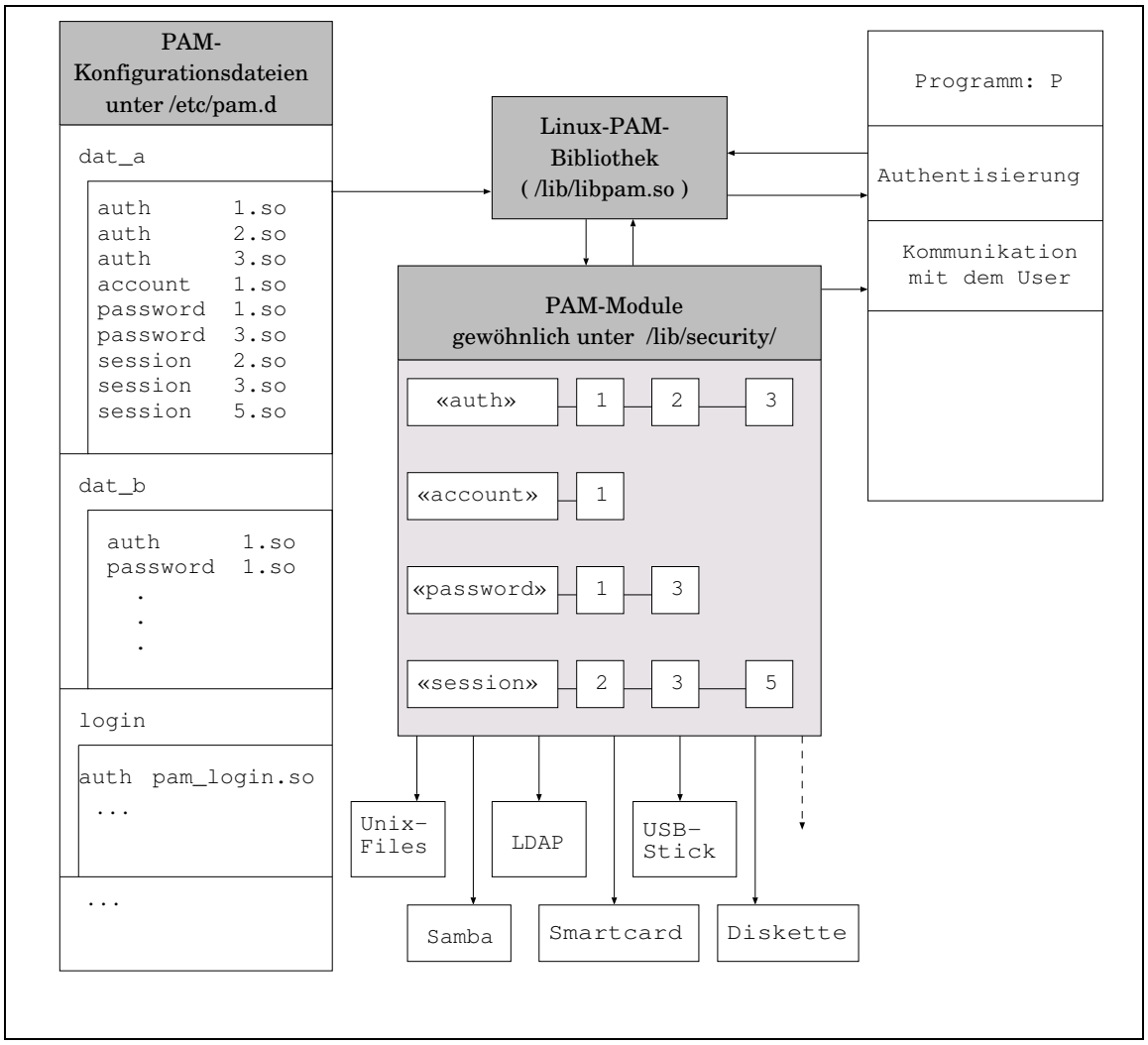



**Abbildung 5.1:** *Das Programm (rechts) authentifiziert seine Benutzer mit Hilfe der PAM-Bibliothek (Mitte). Die Bibliothek liest ihre Konfigurationsdatei z.B.* `/etc/pam.d/login` *(links), lädt die dort aufgeführten Module (*`pam_login.so`*) aus* `/lib/security` *und greift bei Bedarf auf weitere Dienste (unten) wie Samba oder LDAP zu.*

### Aufbau der Konfigurationsdatei

Jede Zeile einer PAM-Konfigurationsdatei entspricht einem Konfigurationseintrag für das zu verwendende Modul und besteht aus 4 Einträgen:

`<Modultyp>` `<Modulsteuerung>` `<Modulpfad>` `<Argumente>`

Als Beispiel wird in der Abbildung 5.2 die Konfigurationsdatei für das Programm "`login`" gezeigt.

Die einzelnen Felder haben folgende Bedeutung:

**Modultyp** Dieser Eintrag bestimmt, welche Managementfunktion ein Eintrag erfüllt. Es gibt die folgenden 4 Modultypen:

- `auth`:
  Das Modul wird zur Authentifikation des Benutzers verwendet.
- `account`:
  Das Modul wird für das Account Managing verwendet und prüft zum Beispiel, ob sich ein Benutzer zu dieser Tageszeit anmelden darf oder noch genügend Rechenzeit auf

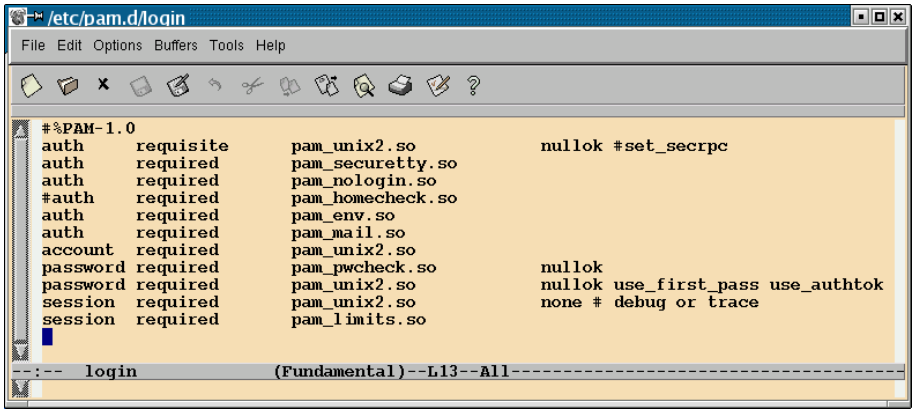



**Abbildung 5.2:** *Konfigurationsdatei* `/etc/pam.d/login` *für das Programm "*`login`*"*

seinem Konto hat.

- `password`:
  Das Modul wird zum Ändern des 'authentication token' (meist ein Passwort) verwendet.
- `session`:
  Das Modul stellt Sessionmanagementfunktionen zur Verfügung, welche unter anderem Umgebungsvariablen setzen oder Verzeichnisse bereitstellen.

In der Konfiguration einer Anwendung können für dieselbe Aufgabe mehrere Einträge (Zeilen) angegeben werden. Die entsprechenden Module werden dann zu einem Stapel zusammengefasst (Modul-Stacking) und in der Reihenfolge ihrer Definition abgearbeitet. Der aufrufenden Anwendung wird am Schluss ein für den ganzen Stapel repräsentatives Endergebnis zurückgegeben und nicht das Resultat jedes einzelnen Moduls.

**Modulsteuerung** Die zweite Spalte legt das Verhalten von PAM fest. Sie bestimmt, wie das Endergebnis des betroffenen Modul-Stapels beeinflusst wird, wenn ein einzelnes Modul mit einer Erfolgs- oder Fehlermeldung endet. Unter bestimmten Bedingungen werden weiter unten stehende Module gar nicht mehr erreicht. Folgende vier Werte sind möglich:

- `required`:
  Jedes als required gekennzeichnete Modul muss erfolgreich durchlaufen werden, um zu einem positiven Gesamtergebnis "`success`" (Erfolg) zu kommen. Sobald ein Modul fehlschlägt, lautet das Resultat "`fail`" (Misserfolg). Dennoch durchläuft PAM die weiteren Module dieser Kategorie.
- `requisite`:
  Auch hier ist der Erfolg des Moduls zwingend für ein positives Endergebnis. Bei einem Misserfolg wird die Verarbeitung allerdings sofort abgebrochen und die folgenden Module nicht mehr ausgeführt.
- `sufficient`:
  Ist das Modul erfolgreich, ist auch das Endergebnis positiv und die Verarbeitung wird abgebrochen. Allerdings dürfen vorher keine Required- oder Requisite-Module fehlgeschlagen sein. Wenn das Modul keine Erfolgsmeldung liefert, wird mit dem nächsten Modul fortgefahren und das Endergebnis bleibt unbeeinflusst.
- `optional`:
  Das Modul wird ausgeführt, aber beeinflusst nicht das Endergebnis. Nur wenn alle anderen Module im Stapel weder Erfolg noch Misserfolg melden, nimmt PAM das Ergebnis des Optional-Moduls. Das kann allerdings bei falscher Anwendung leicht zu

Sicherheitslücken führen.

Falls das Endergebnis nicht eindeutig ist, weil z.B. nur Optional-Module definiert waren, wird der Aufruf als Misserfolg gewertet.

**Modulpfad** Dieser Eintrag legt fest, wo PAM nach der entsprechenden Teilbibliothek sucht. Standardmäßig erwartet es seine Module in "`/lib/security`". In diesem Fall genügt der Modulname. Andernfalls gilt die klassische Unix-Vorgehensweise: Pfadnamen, die mit "`/`" beginnen, werden als absolut interpretiert, und andere als relativ zu "`/lib/security`".

**Argumente** Diese Spalte ist optional und enthält nur bei einigen Modulen einen Eintrag. Die Argumente werden als Liste angegeben, deren Felder durch Leerzeichen getrennt sind.

# Kapitel 6

# Software und Speichermedien

## 6.1 Software zur sicheren Datenkommunikation

Für die Umsetzung des entwickelten Konzeptes (Kapitel 7) konzentriert sich diese Arbeit auf die Verwendung von Open-Source-Software. Das sind Softwareprogramme, deren Quellcode für jeden frei verfügbar und einsehbar ist und jeder Benutzer kann ihn im Prinzip beliebig an seine Bedürfnisse anpassen.
Eventuell sicherheitskritische Funktionen können von jedem Benutzer persönlich kontrolliert und bei Bedarf deaktiviert werden. Als Beispiel für Open-Source-Softwareprojekte ist das Betriebssystem Linux der bekannteste Vertreter.

### 6.1.1 OpenSSL

OpenSSL ist eine Implementierung des SSL[1]/TLS[2]-Protokolls, das von allen gängigen Web-Browsern unterstützt wird und in einer großen Anzahl anderer Anwendungen Einsatz findet. OpenSSL bietet neben zahlreichen Funktionen zur Zertifikat-Verwaltung verschiedene kryptographische Funktionen. Es basiert auf dem SSLeay-Paket, das von Eric A. Young und Tim Hudson entwickelt wurde. OpenSSL als Nachfolger von SSLeay wird derzeit von einer unabhängigen Gruppe weiterentwickelt. [Lit. 8]
Das Paket umfasst mehrere Applikationen, z.B. zur Erzeugung von Zertifikaten (siehe 3.5), Zertifizierungsanträgen und zur Verschlüsselung. Die einzelnen Applikationen sind zusammengefasst in einem Kommandozeilen-Programm: `openssl`.

### 6.1.2 OpenSSH

Secure Shell (SSH) ist ein Programm, mit dem man sich an einem anderen Rechner im Netzwerk einloggen kann, um interaktiv mit Programmen und Daten arbeiten zu können [Lit. 9]. Dabei wird die Kommunikation über eine als unsicher angenommene Verbindung verschlüsselt, damit die Datenübertragung nicht abgehört werden kann. Zusätzlich bietet SSH eine zuverlässige gegenseitige Authentisierung (mit RSA-Verschlüsselung) der Partner (Benutzer und Rechner).
Da die Verschlüsselung der Kommunikation vor der Identifizierung gestartet wird, werden niemals Passwörter oder andere Informationen im Klartext übertragen.

[1] SSL - Abkürzung von engl.: secure socket layer
[2] TLS - Abkürzung von engl.: transport layer security

## 6.2 Speichermedien für Passwörter oder Schlüssel

Ein Speichermedium dient zur Speicherung von Daten bzw. Informationen.
In diesem Kapitel werden ausgehend von den Grundarten der Speicherung die möglichen Speichermedien für die Schlüssel der Kryptoverfahren diskutiert und Vor- und Nachteile aufgezeigt. Die Entscheidung für die ausgewählten Schlüsselspeicher wird an geeigneter Stelle im Kapitel 7 ”Konzept eines sicheren Linux-Zugangs” ausgewertet.

### 6.2.1 Gedächtnis

Einfache Passwörter, Passwörter aus scheinbar zufälligen Buchstabenkombinationen, die zum Beispiel die Anfangsbuchstaben der Wörter eines Satzes sind, oder PIN-Zahlen (als Schlüssel) kann man sich leicht merken.
Das Problem ist zum Einen die Vielzahl solcher Passwörter und Schlüssel, die man sich merken muss, und zum Anderen die Länge der Schlüssel. Die 4-stellige PIN-Nummer von der Bankkarte ist für die meisten noch machbar, doch mit dem Behalten von Telefonnummern, die selten aus mehr als 20 Ziffern bestehen, haben viele Menschen schon Probleme. Die Schlüssel in den erforderlichen Längen für kryptographische Verfahren wie beim RSA-Algorihtmus (Kapitel 3.4), kann man sich aber unmöglich merken.
Doch auch bei kürzeren Passwörtern und Schlüsseln kann es leicht zu Verwechselungen oder zum Vergessen kommen. Daher werden in den meisten Fällen die Passwörter oder Schlüssel aufgeschrieben.

### 6.2.2 Papier

Papier gehört zu den nicht EDV-spezifischen Speichermedien und ist seit seiner Erfindung das klassische Speichermedium zur Aufbewahrung von Wissen und Informationen. Diese können durch den Menschen ohne technische Hilfsmittel unmittelbar verarbeitet werden. Das ist ein Grund, warum Papier für die Speicherung von Passwörtern und Schlüssel ungeeignet ist, da kein Schutz vor unbefugtem Lesen besteht.

Ein weiterer Grund gegen die Wahl als Speichermedium für Schlüssel ist der Aufwand. Schlüssel per Hand korrekt aufschreiben zu wollen, die mehrere hundert Stellen lang sind, und sie fehlerfrei wieder abzulesen und in eine Tastatur einzugeben, ist nicht durchführbar.

### 6.2.3 Disketten

Disketten sind preiswerte magnetische Wechseldatenträger und da traditionell noch die meisten Rechner mit Diskettenlaufwerken ausgerüstet sind, haben sie auch heute noch eine erhebliche, wenn auch abnehmende Bedeutung für die Sicherung und den Austausch von kleineren Datenmengen (primär Textdateien). Disketten sind wiederbeschreibbare und dauerhafte Speichermedien. Heute gängige 3,5 Zoll Disketten (Standarddisketten) besitzen eine Speicherkapazität von 1,44 MB. Da aber die Nachfrage nach Speichermedien mit sehr viel höheren Kapazitäten steigt und die Diskette als primäres Medium für den Datenaustausch nicht mehr ausreicht, wird sie bald weitestgehend von CD-ROM, DVD, USB-Memory-Sticks u.a. abgelöst sein.

Ein Vorteil ist der direkte Zugriff, das heißt, man kann mit Kenntnis der Adresse sofort auf jeden beliebigen Datensatz zugreifen.
Auf einem Linux-System kann man Disketten erst nutzen, wenn man das Laufwerk als Dateisystem einbindet. Der Befehl dazu lautet ”`mount /media/floppy`”. Die Diskette wird wie ein

gewöhnliches Verzeichnis behandelt und der Benutzer kann sich mit den üblichen Linux-Befehlen in den Unterverzeichnissen der Diskette bewegen, Dateien lesen, kopieren, umbennen, löschen usw. Nach der Benutzung löst man die Laufwerkeinbindung mit "`umount /media/floppy`".
Zur Speicherung von Schlüsseln und Passwörtern ist die Kapazität einer Diskette ausreichend.

### 6.2.4 USB-Sticks

USB-Sticks, auch als **Memorysticks** bezeichnet, sind ein modernes elektronisches Speichermedium mit einer USB-Schnittstelle, über die sie im Regelfall an jeden Rechner angeschlossen werden können, der in den letzten Jahren hergestellt wurde.
Die Speicherkapazität reicht derzeit von 128 MB bis 2 GB. Geräte mit weniger als 128 MB sind kaum noch auf dem Markt. Somit sind diese kleinen und leichten Datenspeicher ideal für den Transport größerer Datenmengen.
Auf einem USB-Stick können Standard-Filesysteme (Unix, Windows) abgelegt werden, die mittels Hotplug unmittelbar im System zur Verfügung stehen.
Beachten muss man, dass Datenverluste wahrscheinlich sind, wenn man den Stick abzieht, so lange darauf noch Dateien geöffnet sind. USB-Sticks sind im Vergleich zu Disketten relativ teuer im Anschaffungspreis. Betrachtet man aber den Preis/MB sind sie einer Diskette überlegen. Dennoch gibt es geeignetere Speichermedien für die dauerhafte Sicherung und Archivierung von Daten als Hauptaufgabe.

### 6.2.5 Chipkarten

Chipkarten gehören neben Prägekarten und Magnetstreifenkarten zu den Plastikkarten, die u.a. als Telefonkarten, Kreditkarten, Krankenversicherungskarten und Kundenkarten bekannt sind.
Eine **Plastikkarte** ist ein kleinformatiger viereckiger Datenträger aus Kunststoff, auf dem für Menschen erkennbare und maschinenlesbare Informationen enthalten sein können. Ersteres sind zum Beispiel Schriftfelder und Fotos, zur Speicherung der maschinenlesbaren Informationen können Strichcodes, Schriften und magnetische, optische, sowie Halbleiterspeicher dienen [Lit.10].

**Chipkarten** sind elektronische Datenträger. Sie verwenden Halbleiterbauelemente (Chips) zur Datenspeicherung. Zum Auslesen der Informationen benötigt man einen Kartenleser mit einem passenden Treiber.

#### Arten der Chipkarten

- **Speicherkarten**
  Speicherchipkarten enthalten nur elektronischen Speicher. Bei einfachen Speicherkarten (z.B. Krankenversichertenkarte) kann auf diesen direkt zugegriffen werden. Dieser Kartentyp besitzt keine Sicherheitsmechanismen, man kann die Karte leicht auslesen und beschreiben.
  Bei den intelligenten Speicherchipkarten (z.B. Telefonkarten) ist der Speicher nur über eine fest verdrahtete Sicherheitslogik zugänglich. Die Karte kann leicht ausgelesen werden, doch zum Beschreiben wird meist eine vierstellige PIN benötigt.
- **Chipkarten mit Mikroprozessor**
  Mikroprozessorkarten werden auch als **Smartcards** bezeichnet und sind für vielseitige Anwendungsgebiete einsetzbar. Sie beinhalten neben den Speichereinheiten einen Mikroprozessor und funktionieren wie ein kleiner Computer.
  Im einfachsten Fall enthalten diese Karten genau ein Programm, das auf eine spezielle Anwendung optimiert wurde und nur noch dafür einsetzbar ist.
  Moderne Chipkarten ermöglichen es dem Benutzer, die Karte speziell zu programmieren und

so mehrere, verschiedene Anwendungen zu integrieren. Auf die Daten der Karte kann nur mit Hilfe des Prozessors zugegriffen werden.

- **Kontaktlose Chipkarten**
  Ein großer Nachteil der Chipkarten mit Kontakten ist Verschmutzung des Kontaktfeldes. Kontaktlose Chipkarten besitzen keine mechanischen Kontakte und müssen nicht mehr in einen Kartenleser eingesteckt werden, sondern werden am Empfängergerät (bis zu einer Entfernung von einem Meter) vorbeigeführt. Hauptanwendung für diese Karten sind Zugangskontrollen und im öffentlichen Personenverkehr.

Es gibt drei genormte (nach DIN ISO 7810) Kartenformate (Abbildung 6.1).

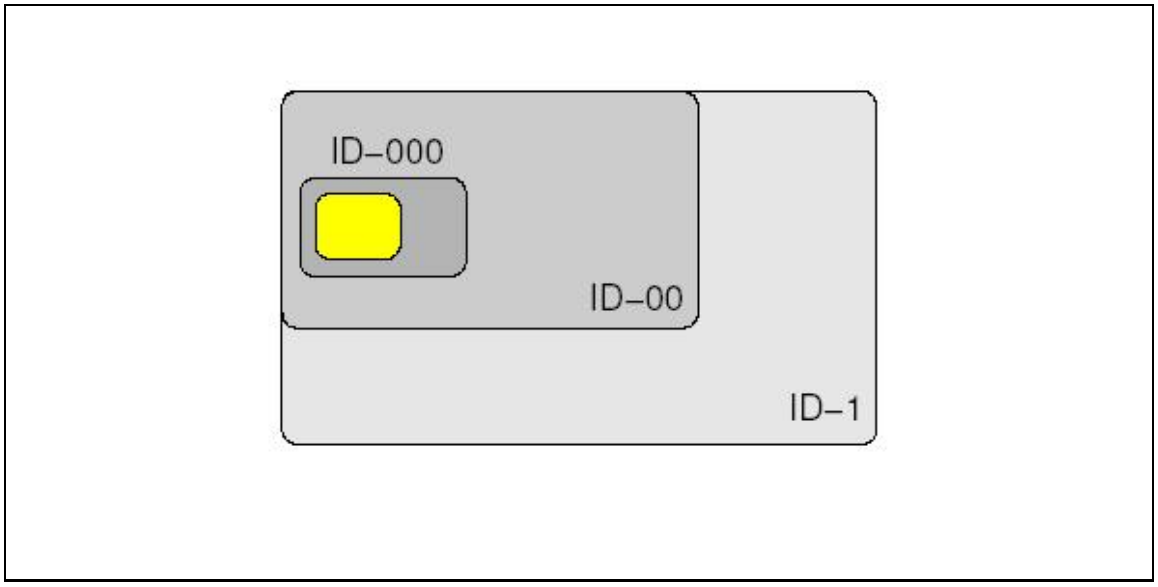



**Abbildung 6.1:** *Standardformate für Chipkarten (ID-1 Kreditkartengröße, ID-000 SIM-Kartengröße in Mobiltelefonsystemen)*

### Smartcards

Der gesamte Datenaustausch zwischen der Smartcard und dem Terminal (Kartenleser) findet über sogenannte Application Protocol Data Units (APDUs) statt. Diese Smartcard-Befehle sind nicht einheitlich und bis auf einige Basisbefehle weichen die Befehlssätze der unterschiedlichen Hersteller voneinander ab.

Die Dateiverwaltung ist hierarisch in Baumstruktur organisiert. Es gibt ein Wurzelverzeichnis (Master File) und bei Smartcards, die über mehr als eine Anwendung verfügen, können noch Unterverzeichnisse (Dedacted Files) angelegt werden, unter denen die verschiedenen Anwendungen liegen. Die eigentlichen Daten sind in den Elementaryfiles abgelegt.
Die Namen der Dateien und Verzeichnisse (Fileidentifier - FID) bestehen aus 2 Byte hexcodierten Zahlen (z.B. 3F00 für das Masterfile). Es gibt einige fest vergebene FIDs (Norm EN 726-3). Abgesehen von diesen können die FIDs frei gewählt werden. Allerdings muss auch hier die Dokumentation der verwendeten Karte beachtet werden, da sich nicht alle Kartenhersteller an die Standards halten und einige Dateinamen für sich reservieren.
Um auf eine Datei zugreifen zu können, muss erst die Anwendung, in der sich die Datei befindet, und dann die Datei selbst selektiert werden. Die Anwendung ist ebenfalls hexcodiert mit 5 bis 15 Byte.

Diese Karten sind prädestiniert für den Einsatz als Zugangssystem für Rechner. Ihr Vorteil liegt in

der einfachen Bedienbarkeit und der hohen Sicherheit gegen Fälschung und unbefugtem Auslesen. Der Nachteil sind die Kosten für die Karte, die im Gegensatz zum USB-Stick nicht zusätzlich zur allgemeinen Speicherung von Daten verwendet werden kann. Außerdem müssen für die Rechner Lesegeräte beschafft werden und deren Treibersoftware muss in das Betriebssystem integriert werden.

# Kapitel 7

# Konzept eines sicheren Linux-Zugangs

## 7.1 Anforderungen und Voraussetzungen

Ein Konzept für ein *Two Factor Authentication System* bietet für den Login-Vorgang an Linux-Netzwerk-Rechnern zusätzliche Sicherheit. Es ist das Ziel, dieses mit wenig Aufwand zu realisieren.
Dabei wird der vorhandene Unix-Login ergänzt, indem das zur Authentisierung benötigte Passwort auf einem persönlichen Datenträger abgelegt und vor Missbrauch geschützt wird. Als mögliche Speicher werden folgende Medien untersucht: Smartcard, Diskette und USB-Stick (siehe Abschnitt 6.2).
Somit benötigt der Benutzer zwei Dinge: Das Wissen des Passwortes und den Besitz von bestimmten Dateien auf seinem persönlichen Datenträger.

Das Konzept soll für ”Kommandozeilen-Login” und ”Fenster-Login” realisiert werden.

#### Auswahl des Speichermediums

Da die Entwicklung eines geeigneten Konzeptes abhängig von dem benutzten Speichermedium für das Passwort und den Schlüssel ist, werden diese als erstes untersucht.

- **Smartcards**
  Zuerst wurde die Verwendung von Smartcards geprüft. Diese zukunftsfähige Chipkarte mit ihrem durch physikalische und logische Sicherungsfunktionen geschützten Speicher und ihren handlichen Abmessungen scheint bestens für eine sichere Speicherung der geheimen Schlüssel für Kryptoverfahren (siehe Kapitel 3) geeignet zu sein.
  An der Züricher Hochschule Winterthur wurde ein Login-Mechanismus mit Smartcards realisiert [Lit. 7], worüber im Linux-Magazin 05/2002 [Lit.11] berichtet wurde.
  Der Vorteil des benutzten Verfahrens ist, dass der private Schlüssel und das Passwort auf die Karte geschrieben werden und nachträglich nicht mehr ausgelesen werden können.

  Es gibt aber auch einige Nachteile für den praktischen Einsatz:
  - Jeder Linux-Arbeitsplatz muss mit einem Kartenleser ausgestattet werden, was mit Kosten verbunden ist.
  - Ein finanzieller Aufwand ergibt sich außerdem durch die Anschaffung der Smartcards,

die die asymmetrische Kryptographie beherrschen müssen und ebenfalls nicht ganz billig sind.

– Man müßte bei der Sorte Smartcards bleiben, für die man sich einmal entschieden hat, da es noch keine Standards hinsichtlich der Dateinamen und der Kommando- und Antwort-APDUs gibt, an die sich alle Firmen halten (siehe 6.2.5). Sonst müssten alle Programme, die von den Smartcards lesen bzw. auf sie schreiben, für jede neue Kartenart im Sourcecode speziell angepasst werden, was dem Ziel, durch die Benutzung von PAM den Programmieraufwand zu senken, entgegensteht.

- **Disketten und USB-Sticks**
  Wie schon unter 6.2.3 beschrieben, haben Disketten den Vorteil, dass noch an fast allen Rechnern Diskettenlaufwerke vorhanden sind und die Anschaffung sehr preiswert ist. USB-Sticks sind in der Anschaffung zwar teurer, kommen aber ohnehin immer häufiger zum Einsatz und der Besitz wird bald Standard sein.
  Außerdem benötigt man zum Beschreiben und Lesen bei diesen Datenträgern keine Kenntnis besonderer Kommandos, sondern man kann die üblichen Linux-Befehle verwenden.

  Der Nachteil bei beiden ist der freie Zugriff. Um zu verhindern, dass auf ihnen gespeicherte Schlüssel und Passwörter unbefugt gelesen oder geändert werden, müssen die Daten verschlüsselt und signiert werden.

Da für das gegebene Umfeld die Vorteile bei Disketten und USB-Sticks überwiegen, wird mit diesen das Konzept entwickelt.

### Voraussetzungen

Von diesen Bedingungen ausgehend, kommt man zu folgenden Voraussetzungen, die für alle zentral betreuten Linux-Rechner zu erfüllen sind:

1. Zentrale Benutzerverwaltung
   Benutzer- und Gruppendaten befinden sich nicht auf jedem einzelnen Rechner, sondern in einem zentralen Verzeichnis, das vom Dispatch verwaltet wird.

2. Nutzung von Linux-PAM
   Im lokalen Betriebssystem die PAM- Schnittstelle für eine Modifikation des Login-Verfahrens benutzt.

3. Alle Linux-Rechner besitzen ein Diskettenlaufwerk oder einen USB-Anschluss,
   da als Speicher eine persönliche Diskette oder ein USB-Memory-Stick zur Anwendung kommt.

Somit sind zwei wesentliche Aufgaben zu erfüllen:

- *Die Erstellung von Schlüsseln und die Erzeugung der entsprechenden Dateien auf dem Schlüsselspeicher.*
- *Die Programmierung eines neuen PAM-Moduls für den Login-Prozess.*

## 7.2 Dispatch-Funktionen

### 7.2.1 Notwendige Programme

Das Dispatch übernimmt, wie in Kapitel 4 beschrieben, das Einrichten und die Betreuung der Benutzer-Accounts und es hat Zugriff auf Einrichtungs- und Bearbeitungsprogramme.
Für die Realisierung der neuen Aufgaben des Dispatch werden eine Reihe von Tools benötigt.

**Einrichtung und Verwaltung der User-Accounts**
Der neue Benutzer muss in der Benutzer-Datenbank bekannt gemacht werden. Es wird für ihn ein RSA-Schlüsselpaar erzeugt und ein Record für die Userdaten ausgefüllt.
Alle Dateien werden vom Dispatch entsprechend verschlüsselt und können dann per E-Mail an den Benutzer versendet werden. Die Dateien am Ort des Dispatch-Systems, die ausschließlich auf dem privaten Datenträger gespeichert werden (Passwortdatei und Datei mit privatem Schlüssel des Benutzers), und die temporären Dateien werden aus Sicherheitsgründen gelöscht.
Die übrigen Benutzerdaten werden in die Verwaltungsstruktur geeignet eingefügt.

**User joinen**
Das Dispatch muss den Benutzer auf jedem zugelassenen Rechner bzw. auf dem Workstation-NIS-Server joinen.

**Schlüsselpaare erzeugen**
Es ist ein Programm nötig, dass die geforderten RSA-Schlüsselpaare erzeugt.

**Ver- und Entschlüsselungsroutinen**
Auf dem Dispatch-Rechner sind spezielle Programme abzulegen, die mit den zuvor erzeugten Schlüsseln Daten und Dateien ver- und entschlüsseln können.

**Check-Routinen**
Bevor ein Benutzer nach der Eingabe seiner Login-Daten zum System zugelassen wird, werden seine Authentisierungsdaten auf Echtheit und Gültigkeit kontrolliert.

### 7.2.2 Einrichten von Benutzer-Accounts

**Benutzer-Daten**

Zusätzlich zu den bisher gespeicherten Daten ist es nötig, auf dem Dispatch-System den öffentlichen Schlüssel des Benutzers abzulegen, der später zur Kontrolle der Signatur nötig ist.
Außerdem kann man in so einem System die gespeicherten Daten leicht um weitere Informationen ergänzen, zum Beispiel solche, die die Autorisierung betreffen, wie
- UserIDs, die der Benutzer zur Anmeldung benutzen darf
- Gültigkeitsdauer
- Systemrechte des Benutzers

In Anlehnung an den Aufbau eines Benutzer-Eintrages (Bild 4.2), könnte die User-Daten-Datei wie folgt aufgebaut sein:

```
Name          | Thomas Mustermann
Institut      | ZAM
Tel           | 1234
Email         | g.mustermann@fz-juelich.de
```

```
Userid        | mustermann xyz123
Host          | zam123 zam234 zam345
gueltig_bis   | 31.12.2004
Optionen      |
pub_modulus   | b9a41338a335e19f3fe225d531e7d91a61ce8aeeb7eb6e371bb3bf45
f80062fba11663f53e8440265b403cb19f781b4814715381c7da7ae4d7807584da1a9d37
pub_exponent  | 010001
```

Neben den persönlichen Daten, die zur Identifikation des Benutzers dienen, enthält die Datei den öffentlichen Schlüssel (Eintrag `'pub_modulus'` und `'pub_exponent'`) des Benutzers. Weitere wichtige Einträge sind: Ablaufdatum und zulässige UserIDs (`'mustermann'` und `'xyz123'`) und die erlaubten Rechner (Eintrag `'Host'`). Die Frage, ob ein Account gültig oder gesperrt ist (z.B Sperrung nach Verlust der Diskette), lässt sich z.B. über das Gültigkeitsdatum oder durch das Entfernen der zulässigen UserIDs klären.

#### Schlüsselpaar erzeugen

Das Dispatch hat außerdem die Aufgabe für sich selbst und für jeden User ein asymmetrisches Schlüsselpaar zu erzeugen. Die dazu notwendigen Routinen sind zu erstellen.

#### Passwort

Im Rahmen der Antragstellung durch den zukünftigen Benutzer (per Online-Antrag über das Web) steht dem Dispatch neben den oben erwähnten Daten auch das vom Benutzer gewählte initiale Passwort zur Verfügung. Dieses kann bei der Erstellung der Authentisierungs-Dateien berücksichtigt werden.

### 7.2.3 Erzeugen und Sichern der Authentisierungs-Dateien

Beim Einrichten des Benutzer-Accounts erstellt das Dispatch die Authentisierungs-Dateien. Die im Folgenden erwähnten Ver- und Entschlüsselungsroutinen sind zu implementieren.

Für den User werden drei Dateien erstellt, die später auf dem persönlichen Datenträger (Schlüsselspeicher) des Benutzers abgelegt werden:

- Privater Schlüssel des Benutzers
- Passwort und
- Benutzer-Daten

Um diese drei Dateien vor dem unbefugten Auslesen und Ändern zu sichern, werden sie mit dem öffentlichen Schlüssel der Benutzerverwaltung verschlüsselt (Schritt (2) in Abb. 7.1).

$$chiffre\_datei = ENCRYPT(PubKey\_Disp, text\_datei)$$

Somit können sie auch nur von der Benutzerverwaltung wieder entschlüsselt

$$text\_datei = DECRYPT(PrivKey\_Disp, chiffre\_datei)$$

und gelesen werden, da nur sie ihren privaten Schlüssel besitzt und dieser auch nicht den Rechner der Benutzerverwaltung verlässt.

Der öffentliche Schlüssel der Benutzerverwaltung ist nicht geheim und jedermann zugänglich. So könnte jeder Angreifer eigene Dateien erstellen und mit $PubKey\_Disp$ verschlüsseln und die Datenintegrität wäre nicht gewährleistet.
Um diesen möglichen Missbrauch zu verhindern, wird zusätzlich das Passwort verwendet. Beim Login-Prozess wird das an dem Client-Rechner eingegebene Passwort mit dem auf dem persönlichen Datenträger hinterlegten verglichen und nur bei Gleichheit der Login gewährt. Deshalb muss das Passwort auf dem Datenträger besonders geschützt sein.
Dazu wird das Passwort vor der Verschlüsselung von der Benutzerverwaltung zusätzlich signiert, das heißt, es wird mit dem privaten Schlüssel des Benutzers verschlüsselt (Schritt (1) in Abb. 7.1).

$$chiffre\_pw = ENCRYPT(PrivKey\_User, passwort)$$

Dieser private Schlüssel des Benutzers ist ebenfalls geheim und kann auch nicht von einem Angreifer ausgelesen werden, da er - wie bereits erwähnt - nur von der Benutzerverwaltung entschlüsselt und gelesen werden kann. Das heißt, mit diesem Schlüssel kann kein gefälschtes Passwort signiert werden.

Somit müsste ein Angreifer ein eigenes Schlüsselpaar erstellen, um das Passwort zu signieren. Um auch das abzufangen, ist der echte öffentliche Schlüssel des Benutzers auf dem Rechner der Benutzerverwaltung hinterlegt und dieser muss zu dem privaten Schlüssel auf dem Schlüsselspeicher passen.
Das heißt, nur wenn sich das Passwort mit dem öffentlichen Schlüssel des Benutzers, der auf dem Dispatch-Rechner abgelegt ist, entschlüsseln lässt, ist das Passwort nicht gefälscht worden.

$$passwort = DECRYPT(PubKey\_User, chiffre\_pw)$$

Lässt sich das Passwort nicht auf diese Weise entschlüsseln, wird der Login-Vorgang abgebrochen.

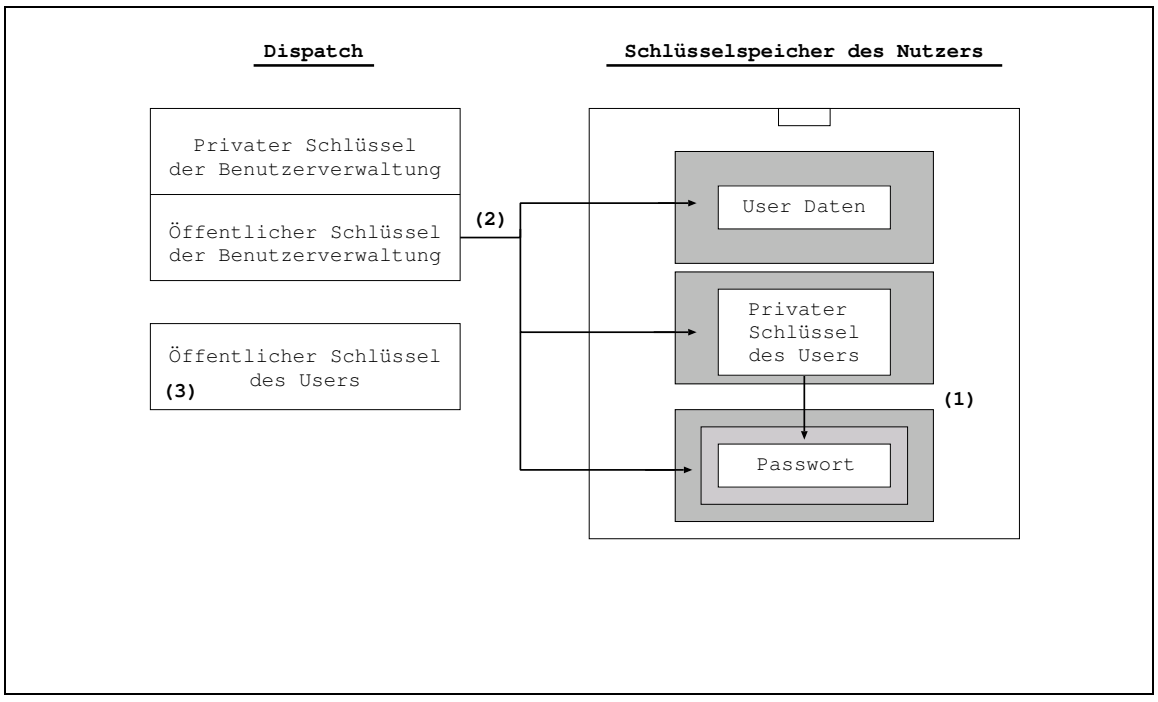



**Abbildung 7.1:** *Erstellen der Authentisierungs-Dateien auf dem Schlüsselspeicher*

Diese Dateien können von der Benutzervewaltung per E-Mail an den Benutzer geschickt werden und dieser speichert sie selbst auf eine Diskette oder einen USB-Stick.

Wenn ein Angreifer die E-Mail abfängt oder mitliest, kann er sie dennoch nicht verwenden, da das hierdrin enthaltene Passwort verschlüsselt ist.

**Der private Schlüssel des Benutzers**

Da man die Echtheit des privaten Benutzer-Schlüssels, mit dem das Passwort signiert wurde, beim Entschlüsselungsversuch feststellt, wird er nur beim Erstellen und Sichern der Passwortdatei benötigt. Während des Login-Prozesses wird er nicht benutzt. Verwendung findet der private Benutzer-Schlüssel erst wieder bei einer möglichen Passwortänderung, dann kann mit ihm das neu gewählte Passwort verschüsselt werden.

## 7.3 Modifizierter Login-Prozess

Um sich als derjenige User zu authentisieren, der er vorgibt zu sein, benötigt der Benutzer seinen Schlüsselspeicher mit den Authentisierungs-Dateien. Nach dem Einlegen der Diskette bzw. Einstecken des USB-Sticks gibt er wie gewohnt seine UserID und sein Passwort über die Tastatur ein.
Die Authentisierungs-Dateien auf dem jeweiligen Datenträger werden zusammen mit dem eingegebenen Passwort und der UserID mittels Secure Copy (`scp`) auf den Rechner des Dispatch übertragen. Dort wird dann mittels `ssh` ein Programm ausgeführt, das die Dateien entschlüsselt und kontrolliert. Wenn alle Kontrollen positiv verlaufen, wird der Login-Prozess fortgesetzt, ansonsten wird der Vorgang mit einer Fehlermeldung abgebrochen.

### 7.3.1 Ablauf einer Benutzeranmeldung

**Das Vorgehen am Client-Rechner:**

- Benutzernamen und Passwort abfragen

  Dazu werden die von der PAM-Bibliothek zur Verfügung gestellten Funktionen aufgerufen

- Schlüsselspeicher ermitteln (USB-Stick oder Diskette)

  Da der USB-Stick der modernere Datenspeicher ist und eher zum Transport von Daten geeignet ist, ist zu erwarten, dass die Benutzer einen USB-Stick als Authentisierungs-Medium verwenden. Außerdem muss ein USB-Stick vor der Benutzung nicht erst gemountet werden. Deshalb wird zuerst kontrolliert, ob ein USB-Stick angeschlossen ist und die nötigen Dateien enthält. Wenn das nicht der Fall ist, wird erst danach auf das Diskettenlaufwerk zugegriffen.

- Kontrollieren, ob die geforderten Dateien auf dem entsprechenden Datenspeicher vorhanden sind

- Aufruf des Check-Programms

  Mittels SSH wird auf dem Dispatch-Server ein Check-Programm aufgerufen und diesem die UserID und das über die Tastatur eingegebene Passwort als Argumente mitgegeben.

- Returnwert

  Über den Returnwert des Check-Programms erhält der Client-Rechner vom Server die Antwort, ob der Login gestattet werden darf.

Der Ablauf ist in der Abbildung 7.2 dargestellt.

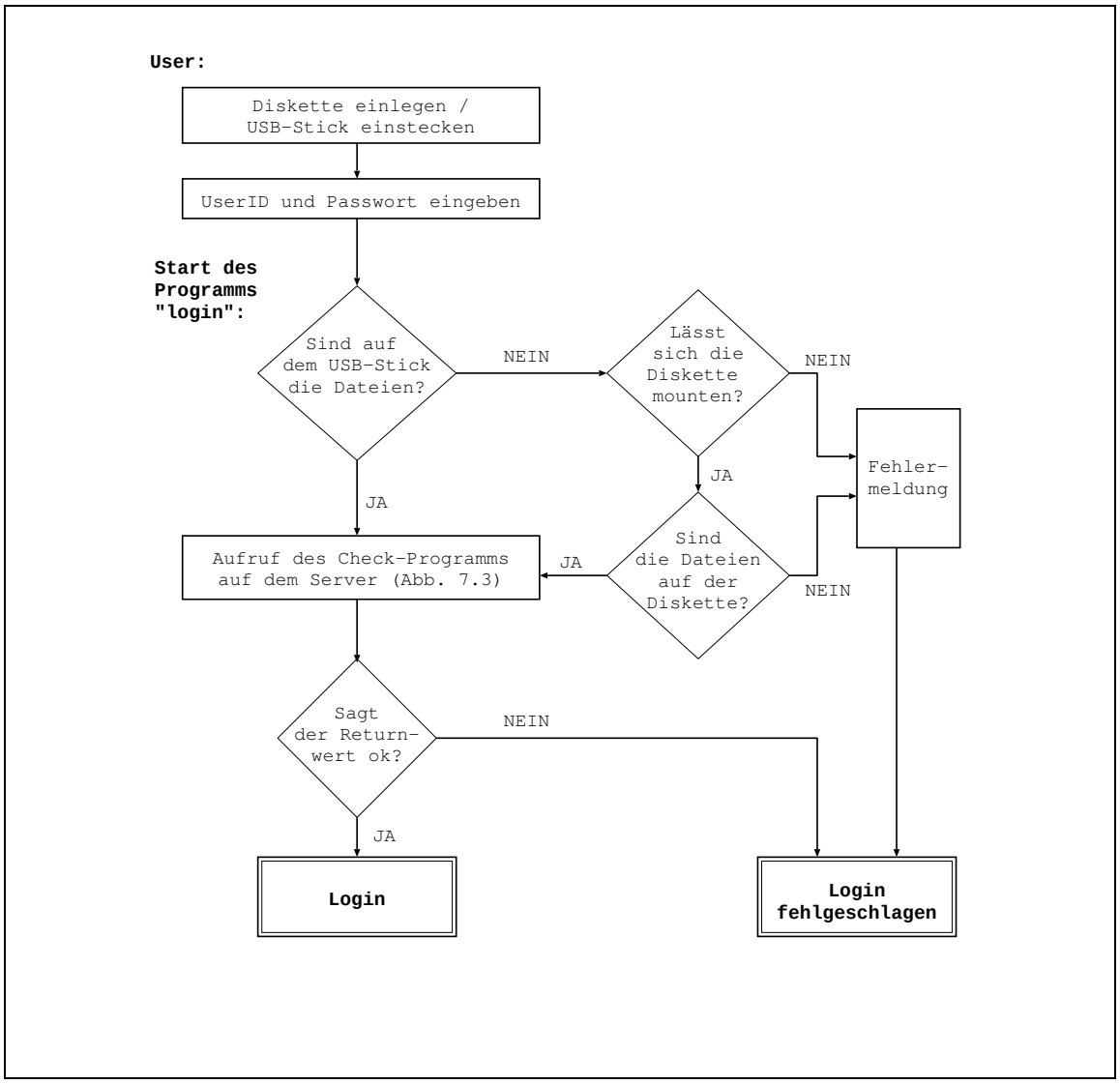



**Abbildung 7.2:** *Ablauf des Login-Vorgangs an dem Client-Rechner*

### Das Vorgehen auf der Serverseite:

Der Server kontrolliert die Authentisierungsdaten auf Echtheit und Gültigkeit.

- Ist der Name des Benutzers (z.B. T.Mustermann) in der Benutzerverwaltung bekannt?

  Das heißt, es wird kontrolliert, ob es bei der Verwaltung einen Eintrag mit dem Namen gibt, wie er in dem verschlüsselten User-Record von dem USB-Stick / von der Diskette steht.
- Darf sich der Benutzer unter der eingegebenen UserID auf diesem Rechner anmelden?
- Sind die Autorisierungsdaten noch gültig?

  Diese Informationen befinden sich in dem User-Record, der beim Dispatch hinterlegt ist. Er kann von dem auf der Diskette gespeicherten abweichen, was zählt, ist was beim Dispatch steht.
- Sind die Autorisierungsdaten nicht manipuliert worden?

  Um eine Manipulation der Authentisierungsdateien zu entdecken, kontrolliert das Programm die Echtheit des privaten Schlüssels des Benutzers, der das Gegenstück zu dem öffentlichen Schlüssel des Benutzers sein muss, welcher beim Dispatch hinterlegt ist.
- Ist der Benutzer der rechtmäßige Besitzer des USB-Sticks / der Diskette?
  Es wird davon ausgegangen, dass nur der rechtmäßige Besitzer das Passwort weiß.

**Ablauf**

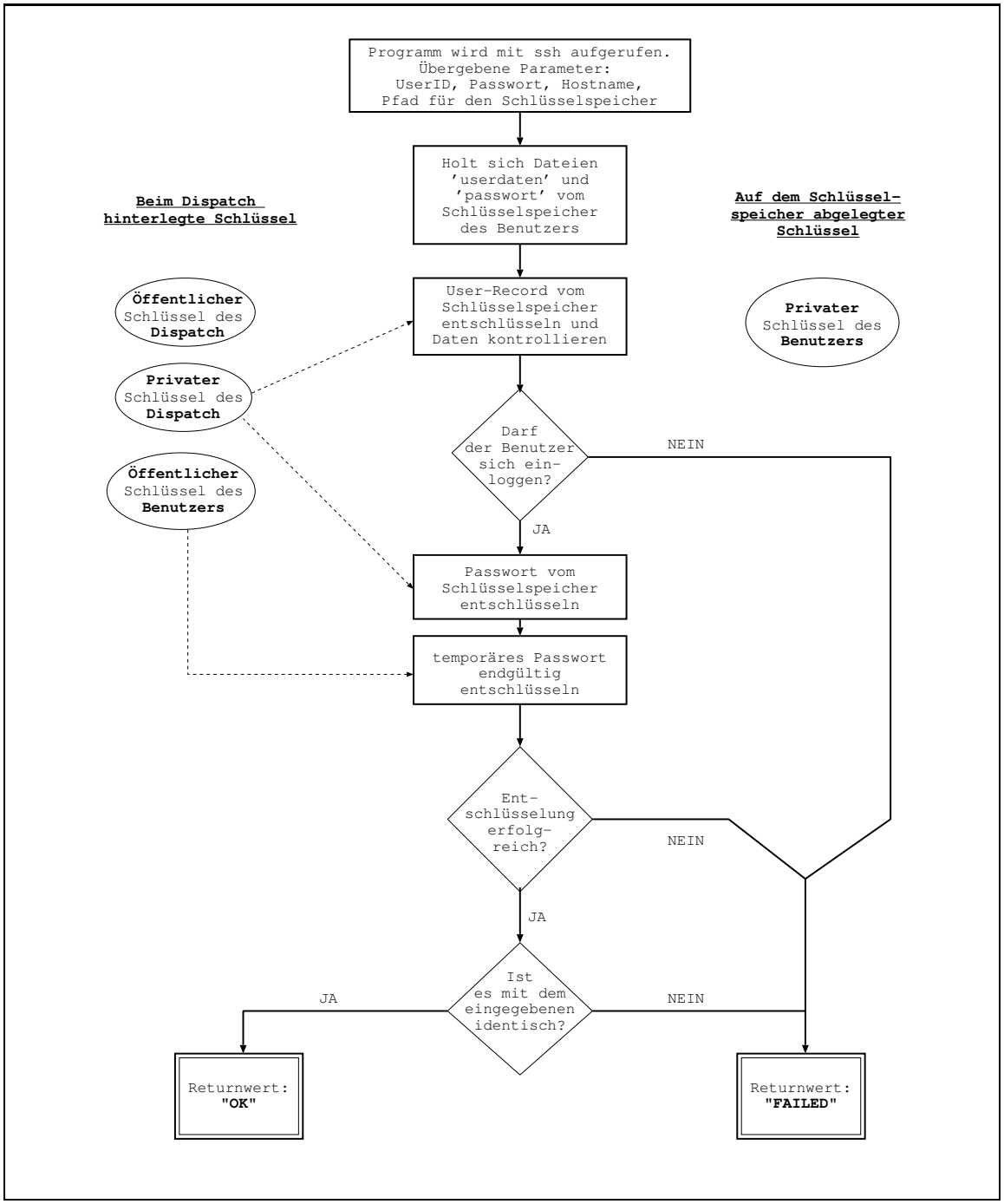



**Abbildung 7.3:** *Ablauf des Login-Vorgangs auf dem Server*

Im einzelnen sieht der Ablauf der Überprüfung folgendermaßen aus (Abb. 7.3):

- User-Daten entschlüsseln und überprüfen

  Der User-Record wird mit dem privaten Schlüssel der Benutzerverwaltung entschlüsselt

  $$record = DECRYPT(PrivKey\_Disp, chiffre\_record).$$

  Er enthält den Namen des Benutzers, einen Eintrag mit den erlaubten UserIDs und das Gültigkeitsdatum (siehe Abschnitt 7.2.2). Somit kann überprüft werden, ob sich der Benutzer mit der eingegebenen UserID einloggen darf bzw. ob die Zugangsberechtigung noch gültig ist.

Ist eines der beiden nicht der Fall oder ist der Name des Benutzers bei der Verwaltung nicht bekannt, dann wird der Login abgelehnt.

- Datei mit dem privaten Schlüssel

  Die Datei mit dem privaten Schlüssel bleibt unbeachtet, da er für den Login-Vorgang nicht benötigt wird.
- Passwort entschlüsseln

  Das Passwort ist doppelt verschlüsselt. Deshalb sind drei Schritte nötig:
  - Passwort mit dem privaten Schlüssel der Benutzerverwaltung entschlüsseln

    $$tmp\_passwort = DECRYPT(PrivKey\_Disp, chiffre\_pw)$$

  - Das temporäre Passwort mit dem öffentlichen Schlüssel des Benutzers entschlüsseln, der beim Dispatch hinterlegt ist.

    $$passwort = DECRYPT(PubKey\_User, tmp\_passwort)$$

    Lässt sich das temporäre Passwort nicht entschlüsseln, wurde es mit einem falschen privaten Schlüssel verschlüsselt und der Login-Prozess wird abgebrochen.

  - Vergleich des eingegebenen Passwortes mit dem entschlüsselten aus der Authentisierungsdatei: Sind die beiden gleich, wird der Login gewährt, ansonsten wird der Zugang zum System verweigert
- Login erfolgreich

  Sind alle Kontrollen positiv verlaufen, müssen alle temporären Dateien wieder gelöscht werden, erst dann kann der Benutzer zum System zugelassen werden.
- Verhalten im Fehlerfalle

  Sollte während der Kommunikation mit dem USB-Stick/mit der Diskette oder mit dem Server ein Fehler auftreten, oder bei der Kontrolle der Authentisierungsdateien festgestellt werden, dass der Benutzer sich nicht einloggen darf, müssen auch in diesem Fall vor Beendigung des Programms alle temporären Dateien gelöscht werden.

### 7.3.2 Das PAM-Modul `pam_login`

Die Anwendung (also das Login-Programm), muss mit der Bibliothek libpam verlinkt sein. Da das derzeitige System bereits auf Linux-Pam aufbaut, muss nur das Modul `pam_login` (wird für den Kommandozeilen- und den Fenster-Login verwendet) angepasst werden.
Für die Aufgaben der PAM-Module stehen verschiedene Funktionen zur Verfügung:

- Authentisierung (auth)

  `pam_sm_authenticate()` authentisiert den Benutzer zum Beispiel durch Abfragen eines Passwortes.
- Account Management (account)

  `pam_sm_acct_mgmt()` überprüft, ob sich ein Benutzer überhaupt anmelden darf oder ob z.B. schon eine maximale Anzahl Benutzer angemeldet sind.
- Session Management (session)

  `pam_sm_open_session()` wird zum Öffnen einer Benutzer-Session aufgerufen und `pam_sm_close_session()` wird zum Schließen einer Benutzer-Session aufgerufen.

- Passwort-Management (password)
  `pam_sm_chauthok()` dient dem Ändern des authentication token (Passwort o.ä.).

In der Funktion `pam_sm_authenticate()` müssen die Aufgaben des Servers (vergl. den im vorigen Abschnitt beschriebenen Ablauf) einprogrammiert werden. Dadurch wird das bisherige Vorgehen (Vergleichen des eingegeben Passwortes mit dem in der Datei `/etc/shadow` gespeicherten - siehe Abschnitt 5.1 auf Seite 19) ersetzt.
Die anderen Funktionen werden für den Login-Prozess nicht benötigt.

## 7.4 Authentisierungsserver

Auf dem Dispatch-Server, wo die eigentliche Authentisierung stattfindet, wird eine sogenannte Restricted Shell verwendet, das heißt, unter dieser ist das Absetzen nur bestimmter Befehle erlaubt. Die Ausführung anderer Befehle ist nicht möglich.
Dieses Vorgehen erlaubt es dem Login-Prozess, von dem Rechner aus, an dem sich der Benutzer einloggen möchte, über `ssh` (siehe 6.1.2) die Ausführung des Kontrollprogramms auf dem Dispatch-Server anzustoßen. Dazu müssen die öffentlichen ssh-Schlüssel jedes Client-Rechners beim Dispatch hinterlegt sein (siehe Abschnitt 8.5 im Kapitel ”Implementierung”).

# Kapitel 8

# Implementierung

## 8.1 Verwendete Programmier-Sprachen

- **C**
  Die PAM-Module sind eine Sammlung von C-Programmen, deshalb wird auch das modifizierte Modul `login` in C programmiert.
  Die Programme zum Ver- und Entschlüsseln und zur Schlüsselerzeugung nutzen Funktionen aus der OpenSSL-Library, welche ebenfalls in C programmiert wurde. Daher werden auch diese Programme mit C realisiert.
- **Perl**
  Perl ist eine Interpreter-Sprache, mit der man leicht Texte verarbeiten kann. Es bietet eine hohe Geschwindigkeit bei Suchfunktionen und die regulären Ausdrücke sind sehr effizient implementiert. Außerdem kann mittels Handles auf die Ein- und Ausgabe von Programmen zugegriffen werden. Diese Funktionen sind für das Check-Programm auf dem Server sehr nützlich, da es die Kontrolle der Authentisierungs-Daten übernimmt. Dabei werden Textdateien durchsucht, wobei besonders die regulären Ausdrücke hilfreich sind. Deshalb wurde es in Perl realisiert: `check_login.pl`.
- **Shell-Skript-Sprache ksh**
  Damit die Befehle für die Erstellung der Benutzer-Accounts und -Daten nicht alle nacheinander per Hand eingegeben werden müssen, wurde der Vorgang in einem ksh-Skript zusammengefasst: `gen_user`.

## 8.2 Routinen zur Schlüsselerzeugung, Ver- und Entschlüsselung

### 8.2.1 Allgemeines

Die OpenSSL-Library bietet für diese Zwecke Funktionen an. Die wichtigsten sind in der Tabelle 8.1 aufgelistet.

Alle folgende beschriebene Programme befinden sich unter der UserID `slnl` (siehe 8.3.1 und 8.4.1) auf dem Dispatch-Rechner im Unterverzeichnis `$HOME/bin`. Sie bilden ein Commandline-interface zu den Funktionen der RSA-Bibliothek von `openssl`.

| RSA_new | Anlegen und Initialisieren einer RSA-Struktur |
|---|---|
| RSA_free | Freigabe der RSA-Struktur |
| RSA_generate_key | erzeugt ein RSA-Schlüsselpaar |
| RSA_public_encrypt | Verschlüsseln eines Textes |
| RSA_private_decrypt | Entschlüsseln eines Textes |
| RSA_private_encrypt | Signieren eines Textes |
| RSA_public_decrypt | Verifizieren eines signierten Textes |

**Tabelle 8.1:** *Kryptographische Funktionen, die vom OpenSSL-Paket zur Verfügung gestellt werden*

### 8.2.2 Routine zur Schlüsselerzeugung (`gen_keys`)

Dieses Programm erzeugt ein RSA-Schlüsselpaar und wird einmalig für das Dispatch und bei der Einrichtung jedes neuen Benutzers ausgeführt. Dazu wird die OpenSSL-Funktion `RSA_generate_key` [Lit.13] verwendet.
Die erzeugten Schlüssel werden in der jeweiligen Datei `'private_key'` bzw. `'public_key'` in der abgebildeten Form (siehe Abb. 8.1) abgespeichert.

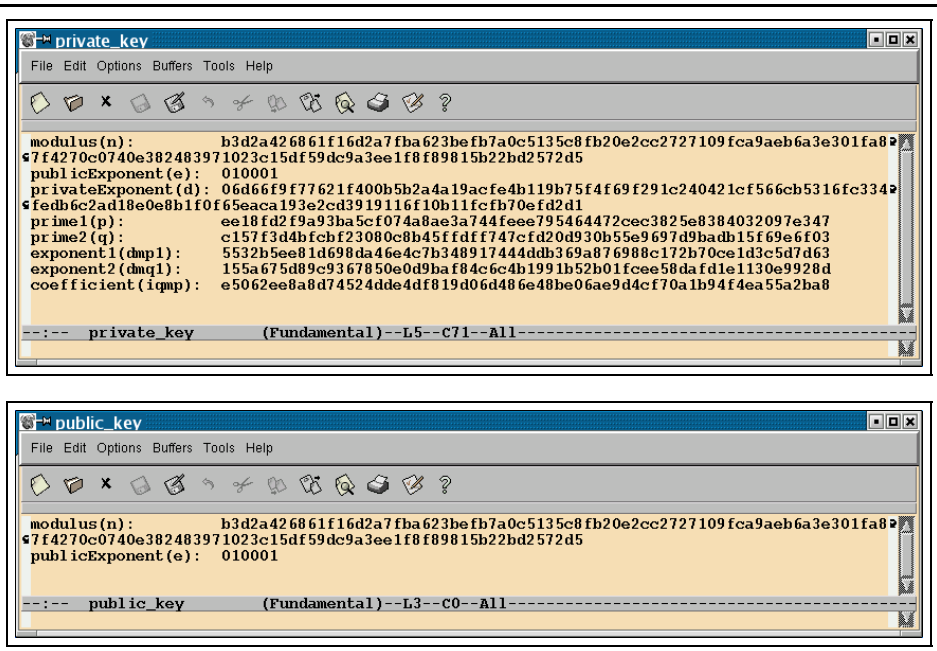



**Abbildung 8.1:** *Struktur der Dateien* `'private_key'` *und* `'public_key'`

### 8.2.3 Verschlüsselungsroutinen

Für die nötigen Verschlüsselungsvorgänge werden die OpenSSL-Funktionen `RSA_private_encrypt` und `RSA_public_encrypt` verwendet.
Dadurch ergaben sich aber Probleme, denn der entstandene Chiffretext (Datei <chiffre_file.pem>) beinhaltete nicht lesbare Zeichen und diese konnten auch nicht einheitlich dargestellt werden. In der Abbildung 8.2 wird einmal die Ausgabe des Chiffretextes in einer Shell und die Ausgabe des selben Chiffretextes mit einem Editor gezeigt.

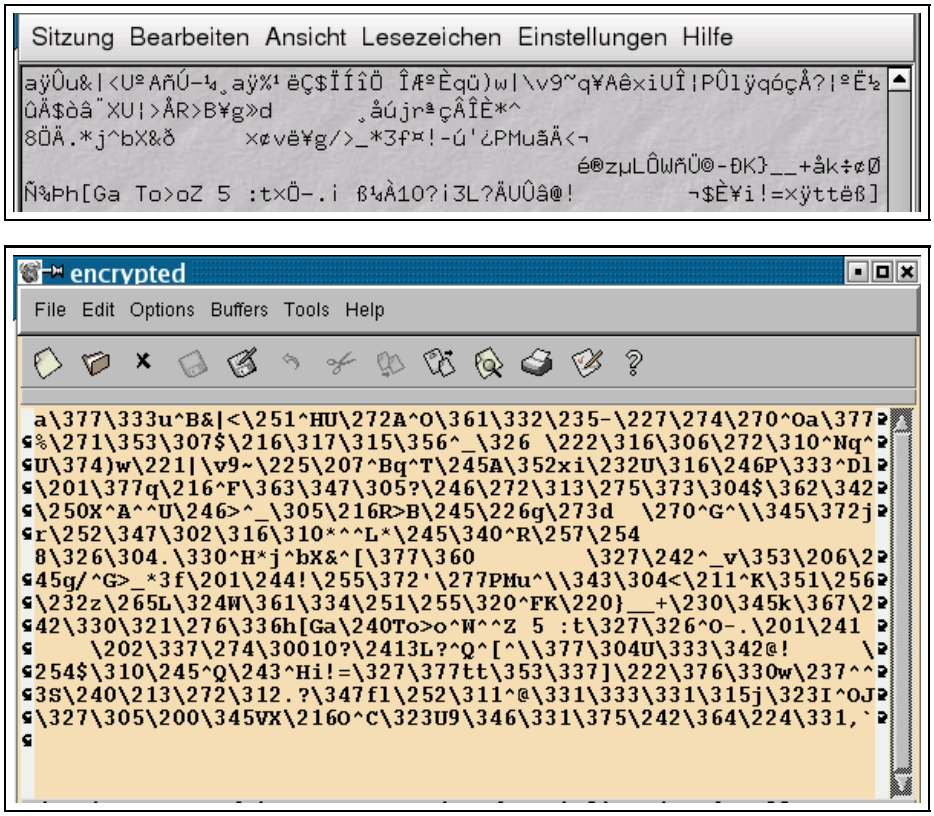



**Abbildung 8.2:** *Ausgabe des Chiffretextes in einer Shell (oben) und im Editor Emacs (unten)*

Es gab viele Beispiele, bei denen eine Weiterverabeitung (Rückverschlüsselung bzw. eine weitere Verschlüsselung) nicht korrekt funktionierten, da die enthaltenen Zeichen teilweise von der Shell interpretiert wurden (z.B. als Dateiende - deshalb ist die Ausgabe in der oberen Abbildung viel kürzer als die Darstellung des selben Chiffre-Textes mit dem Emacs). Auch das Abspeichern der chiffrierten Texte in Strings anstatt in Dateien brachte keine Abhilfe.

### Base64-Konvertierung

Um die Umsetzung des Konzeptes dennoch weiter verfolgen zu können, wird nach jedem Verschlüsselungsschritt mit `RSA_private_encrypt` bzw. `RSA_public_encrypt` der entstandene Chiffretext in das Base64-Format umgewandelt (siehe Abb. 8.3). Dieses Format enthält nur menschenlesbare Zeichen und bereitet keine Probleme bei der Weiterverarbeitung. Es handelt sich dabei um eine Umwandlung anhand einer Tabelle, die nur 64 Zeichen verwendet ($A-Z$, $a-z$, $0-9$, $+$ und $/$). Dazu werden jeweils drei Bytes des Textes in vier Blöcke zu je 6 Bit aufgeteilt, die $2^6 = 64$ mögliche Zeichen ergeben. Jeder Block wird in die entsprechende Zahl (0 bis 63) umgewandelt und den Zeichen der Tabelle zugeordnet. Fehlende Zeichen am Ende werden werden mit $=$ aufgefüllt.

Die Funktion für die Umwandlung der Chiffre-Datei (`chiffre-file.pem`) wird ebenfalls vom OpenSSL-Paket zur Verfügung gestellt.

Der Befehl für das Umwandeln lautet

```
openssl base64 -in <chiffre-file.pem> -out <chiffre-file.b64>
```

Für das Dekodieren des Base64-Formates wird folgender Befehl benutzt

```
openssl base64 -d -in <chiffre-file.b64> -out <chiffre-file.pem>
```

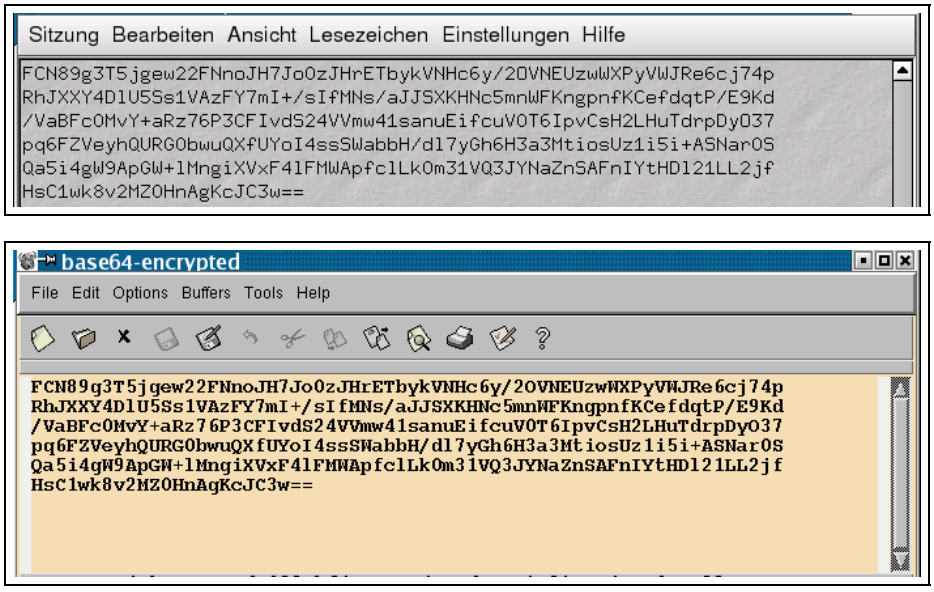



**Abbildung 8.3:** *Die Ausgabe des base64-codierten Chiffretextes in einer Shell (oben) oder im Editor Emacs (unten) ergibt keine Unterschiede*

### Das Programm `encrypt_priv`

Dieses Programm verschlüsselt mit einem privaten Schüssel eine Datei und gibt das Ergebnis in eine Datei aus. Als Argumente werden die Dateinamen in der festgelegten Reihenfolge übergeben. Der Aufruf lautet:

```
encrypt_priv <private key file> <text file> <chiffre file>
```

Das Programm hat folgenden Ablauf:

- Anlegen einer RSA-Struktur:

  ```
  RSA *priv_rsa;
  priv_rsa = RSA_new();
  ```

- Einlesen des privaten Schlüssels (zeilenweise - siehe Schlüssel-Datei in der Abb. 8.1) aus der im ersten Argument angegebenen Datei

  ```
  len = 64;
  for (i = 0; i < len; i++)
    {fscanf(findat,"%2x",&buf[i]);}
  priv_rsa->n = BN_bin2bn(buf, len, NULL);
  fscanf(findat,"\n");
  ```

- Blockweises Verschlüsseln des Textes aus der im zweiten Argument übergebenen Datei:

  Die Klartext-Blöcke müssen kleiner sein als die entstehenden Chiffre-Blöcke. Für eine detailliertere Beschreibung der verwendeten Funktion `RSA_private_encrypt` wird auf die man-Pages verwiesen [Lit.14].

```
/* Verschluesseln in Einheiten zu 50 , Entschluesseln mit 64 Zeichenblock */
  t_buf = 50;
  c_buf = 64;

  /* volle Blöcke */
  count = t_len / t_buf;
  for (i=0; i<count; i++) {
    num = RSA_private_encrypt(t_buf, &textbuf[i*t_buf], &cryptbuf[i*c_buf],
                              priv_rsa, RSA_PKCS1_PADDING);
  }
  /* letzter, kleinerer Block */
  c_len = i*c_buf;
  count = t_len % t_buf;
  if ( count != 0) {
    num = RSA_private_encrypt(count, &textbuf[i*t_buf], &cryptbuf[i*c_buf],
                              priv_rsa, RSA_PKCS1_PADDING);
    c_len = c_len+num;
  }
```

- Zusammensetzen des Chiffretextes und Schreiben in eine Datei
- Lässt sich eine der drei Dateien nicht öffnen bzw. gibt es Fehler beim Einlesen des Schlüssels (z.B. die angegebene Datei enthält keinen privaten Schlüssel) oder beim Verschlüsseln, so wird die entsprechende Fehlermeldung am Bildschirm ausgegeben.

Dieses Programm wird vom Programm `gen_user` (siehe 8.3.1) im Rahmen der Erstellung der Daten für einen neuen Benutzer aufgerufen.

#### Das Programm `encrypt_pub`

Dieses Programm wird ebenfalls von `gen_user` aufgerufen. Es verschlüsselt mit einem öffentlichen Schlüssel und funktioniert ansonsten analog zum Verschlüsseln mit einem privaten Schlüssel.

Der Befehl für den Aufruf lautet:

```
encrypt_pub <public key file> <text file> <chiffre file>
```

Verschlüsselt wird mit der Funktion `RSA_public_encrypt` [Lit.15], die Blockgröße ist ebenfalls 50 Zeichen für den Klartext und 64 Zeichen für den Chiffretext.
Es werden ebenso - wenn nötig - Fehlermeldungen erzeugt und das Ergebnis wird in die durch das dritte Argument festgelegte Datei geschrieben.

### 8.2.4 Entschlüsselungsroutinen

Die Programme zur Entschlüsselung der mit `encrypt_priv` und `encrypt_pub` erzeugten Dateien werden von dem Kontroll-Programm `check_login.pl` (siehe 8.4.2) aufgerufen. Beide Programme haben dieselben Argumente wie die Verschlüsselungsprogramme. Das dritte Argument ist hier optional, das heißt, wenn keine Ausgabedatei angegeben wird, wird der entschlüsselte Klartext auf STDOUT ausgegeben.

#### Das Programm `decrypt_pub`

Dieses Programm ist das Gegenstück zu `encrypt_priv` und verwendet für das Entschlüsseln den öffentlichen Schlüssel aus der Datei, die mit dem ersten Argument angegeben wird.
Der Aufruf lautet:

```
decrypt_pub <public key file> <chiffre file> [opt: text file]
```

Der Ablauf ist ähnlich dem Verschlüsseln. Der Chiffre-Text muss wieder in Blöcke geteilt werden, die in dem Fall 64 Zeichen umfassen. Das OpenSSL-Paket stellt dazu die Funktion `RSA_public_decrypt` zur Verfügung (siehe [Lit.14]).

```
/* Verschluesselt wurde in Einheiten zu 50 => Entschluesseln mit 64 Zeichenblock */
  t_buf = 50;
  c_buf = 64;

  /* count mal ganze Einheiten */
  count = (c_len) / c_buf;
  for (i=0; i<count; i++) {
    num = RSA_public_decrypt(c_buf, &cryptbuf[i*c_buf], &textbuf[i*t_buf],
                             pub_rsa, RSA_PKCS1_PADDING);
    }
```

Im Fehlerfalle, z.B. wenn sich die Chiffre-Datei nicht mit dem in der Datei angegebenen Schlüssel entschlüsseln lässt, wird eine entsprechende Fehlermeldung auf STDOUT ausgegeben.

#### Das Programm `decrypt_priv`

Dieses Programm ist das Gegenstück zu `encrypt_pub` und entschlüsselt die chiffrierte Datei mit einem privaten Schlüssel. Die verwendete OpenSSL-Funktion ist `RSA_private_decrypt` (siehe [Lit.15]).

## 8.3 Benutzer-Administration

### 8.3.1 Erstellung der Benutzer-Dateien

Die für den modifizierten Login-Vorgang nötige Benutzerverwaltung ist derzeit noch nicht an das bestehende Dispatch-System angebunden, daher wird eine vereinfachte Datei-Struktur unter der UserID `slnl` benutzt, die folgendermaßen aufgebaut ist:

```
$HOME:              $HOME/Users:
   Keys                A.Trensch
   Users               T.Mustermann
   bin                 ...
                       user_temp
```

In dem Verzeichnis `Keys` liegt das Schlüsselpaar des Dispatch und unter `bin` sind die ausführbaren Programme abgelegt. Die Benutzer-Verzeichnisse (`T.Mustermann`) enthalten den User-Record (7.2.2) mit den individuellen Benutzerdaten und `user_temp` ist eine Vorlage (Template) für diesen Record.

Mit folgenden Befehlen, die in einem Shell-Skript zusammengefasst werden, erstellt man die Einträge für den neuen Benutzer:

```
cd Users
mkdir <V.Nachname>
cd <V.Nachname>
cp ../user_temp user_daten
~/bin/gen_keys
```

Durch das `gen_keys`-Kommando werden zwei Dateien erstellt: `'private_key'` mit dem privaten Schlüssel und `'public_key'`, die den öffentlichen Schlüssel enthält.
Die fehlenden Daten werden in das Template `'user_daten'` eingetragen und die letzten beiden

Zeilen werden aus der Datei `'public_key'` übertragen.

Außerdem muss das Passwort in eine temporäre Datei `'passw'` geschrieben werden, damit es zusammen mit den anderen Dateien verschlüsselt werden kann.
Diese Befehle zur Verschlüsselung der Dateien können ebenfalls in einem Shell-Skript zusammengefasst werden:

#### Das Programm `gen_user`

Das Programm `gen_user` verschlüsselt die drei Dateien `'passw'`, `'user_daten'` und `'private_key'` und wandelt sie jeweils, wie schon beschrieben (siehe 8.2.3), in das base64-Format um.

```
#! /bin/ksh

# Passwort signieren mit User-Private-Key
~/bin/encrypt_priv private_key passw passwd1pem;

# Signiertes Passwort in base64-Format konvertieren
openssl base64 -in passwd1pem -out passw1pem.b64;

# Passwort verschluesseln mit Dispatch-Public-Key
~/bin/encrypt_pub ~/Keys/public_key passw1pem.b64 passw.b64.pem;

# zur besseren Lesbarkeit wieder in base64-Format konvertieren

openssl base64 -in passw.b64.pem -out  passw.b64.pem.b64;

##########

# Private Key des Users verschluesseln mit Dispatch-Public-Key
~/bin/encrypt_pub ~/Keys/public_key private_key privkey.pem;

# und in base64-Format konvertieren
openssl base64 -in privkey.pem -out privkey.pem.b64;

##########

# User-Daten verschluesseln mit Dispatch-Public-Key
~/bin/encrypt_pub ~/Keys/public_key user_daten userdat.pem;

# und in base64-Format konvertieren
openssl base64 -in userdat.pem -out userdat.pem.b64;

rm passw passwd1pem passw1pem.b64 private_key public_key;
```

Nachdem die verschlüsselten Dateien erstellt wurden und per Mail an den Benutzer verschickt werden können, werden alle nicht benötigten temporären Dateien aus Sicherheitsgründen gelöscht.

### 8.3.2 Benutzer-Record `user_daten`

Der User-Record (7.2.2) wird zum einen verschlüsselt auf den USB-Stick bzw. die Diskette geschrieben. In diesem Record bestimmt die erste Zeile, welches Benutzerverzeichnis für die späteren Kontrollen verwendet wird.

Das User-Verzeichnis hat in diesem Fall den Namen:

```
$HOME/Users/T.Mustermann
```

In diesem Verzeichnis ist der Benutzer-Record ebenfalls hinterlegt. Er wird für die Vergleiche zur Authentisierung und Autorisierung herangezogen. In ihm können Daten und Rechte hinzugefügt und gelöscht werden, ohne dass der Record erneut verschlüsselt, an den Benutzer geschickt und auf dem persönlichen Datenträger abgespeichert werden muss.
Dadurch kann ein Benutzer leicht gesperrt werden, wenn er seinen Datenträger als Verlust meldet. Zum Beispiel könnten seine UserIDs gelöscht werden oder es könnte ein Flag gesetzt werden, dass dieser Datenträger ungültig ist.

## 8.4 Realisierung des Login-Vorgangs

### 8.4.1 PAM: Die C-Programmbibliothek `pam_slnl.so`

Die Abkürzung `slnl` steht für `secure linux network login`.
Für die Modifikation wurde das von [Lit. 7] veröffentlichte PAM-Modul herangezogen und für die Umsetzung des Konzeptes angepasst. Dazu wurde in der Funktion `pam_sm_authenticate` das Einlesen der UserID und des Passwortes (die von dem Benutzer über die Tastatur eingegeben wurden) übernommen. Für diese Aufgabe stellt Linux-PAM die Funktionen `pam_get_user` und `pam_get_item` zur Verfügung. Eine detaillierte Beschreibung dazu befindet sich unter [Lit.12]. Die implementierte Kontrolle des eingegebenen Passwortes wurde ersetzt durch eine eigene Funktion `get_auth_infos`, die den Zugriff auf den persönlichen Datenspeicher des Benutzers für die Passwortkontrolle regelt.

Prototyp der Funktion:

```
int get_auth_infos (char* userid, char* password);
```

Diese Funktion mountet den USB-Stick oder die Diskette als Dateisystem, kontrolliert, ob die Authentisierungsdateien vorhanden sind und stößt auf dem Dispatch-Server die Ausführung des Kontrollprogramms an.

Die Dateien auf dem Schlüsselspeicher haben festgelegte Namen:

| | | |
|---|---|---|
| `passw.b64.pem.b64` | $->$ | Passwort |
| `privkey.pem.b64` | $->$ | Privater Schlüssel des Benutzers |
| `userdat.pem.b64` | $->$ | Record mit den Benutzer-Daten |

Die Endung '`pem`' steht für die Verschlüsselung mit den OpenSSL-Routinen, die Endung '`b64`' steht für die Base64 Konvertierung (siehe Abschnitt 8.2.3).
Die Datei `privkey.pem.b64` wird für die Authentisierung nicht benötigt. Er kann aber für eine spätere Passwortänderung verwendet werden.

Der Aufruf des Kontroll-Programms auf dem Authentisierungs-Server erfolgt über `ssh`. Als Argumente werden die eingegebene UserID (z.B. `xyz123`), das eingegebene Passwort, der Pfadname der Authentisierungsdateien (USB-Stick oder Diskette) und der Hostname (z.B. `zam456`), an dem sich der Benutzer einloggen möchte, mit übergeben.

```
ssh -x server_id@server_host $HOME/check_login.pl \
    $userid $password $path $host > /var/log/slnl/check_login_xyz123.zam456
```

Bei dem Einwählen auf dem Dispatch-Server mit ssh wird eine sogenannte Restricted Shell verwendet, das heißt es ist nur der Aufruf des Programmes `check_login.pl` erlaubt. Der Aufruf

anderer Kommandos oder Programme führen zu einem Fehler.

Die Ausgaben des gerufenen Programms `check_login.pl` werden über die Ausgabeumleitung in eine entsprechende Log-Datei (hier: `/var/log/slnl/check_login_xyz123.zam456`) geschrieben. Diese enthält in der letzten Zeile `SUCCESS` oder einen Fehlercode `FAILED` *xxx*. Das PAM-Modul gibt daraufhin das entsprechende Ergebnis `PAM_SUCCESS` für eine erfolgreiche bzw. `PAM_AUTH_ERR` für eine missglückte Authentisierung zurück. In der jeweiligen Datei `/var/log/slnl/check_login_*` kann der Administrator anhand des Fehlercodes erkennen, woran der Login-Versuch gescheitert ist, ob jemand mit einem falschen Authentisierungsdatenträger oder einem falschen Passwort versucht hat, sich einzuloggen, oder ob ein Fehler im System bestand.
Bei einem fehlgeschlagenen Login-Versuch bekommt der Benutzer nur das Ergebnis mitgeteilt und nicht den Grund, damit es Unbefugten erschwert wird, die richtige Passwort-UserID-Kombination herauszufinden.

Das ausführbare PAM-Modul wird als `/lib/security/pam_slnl.so` gespeichert.

#### Die PAM-Konfigurationsdatei

In der bestehenden PAM-Konfigurationsdatei `/etc/pam.d/login` (siehe Abbildung 5.2 auf Seite 23) wird die erste Zeile durch folgende ersetzt:

```
auth     required     pam_slnl.so
```

Für den Fenster-Login wird die erste Zeile der Datei `/etc/pam.d/xdm` durch dieselbe Zeile ersetzt.

### 8.4.2 Perl-Routine `check_login.pl`

Dies ist das Kontroll-Programm, welches auf dem Dispatch-Rechner läuft und die Authentisierungsdaten überprüft. Als Argumente werden dem Programm die vom Benutzer eingegebene UserID, das eingegebene Passwort, der Pfad des Schlüsselspeichers und der Hostname des Rechners, an dem der Benutzer sich einloggen möchte, übergeben.

Der Ablauf des Programms ist der folgende:

- Die Dateien werden von dem Schlüsselspeicher zum Dispatch-Server kopiert. Dort werden der Eindeutigkeit halber die UserID (z.B. `xyz123`) und der Hostname (z.B. `zam456`) an den Namen angehängt. Die Datei `userdat.pem.b64` heißt dann auf dem Server `userdat.pem.b64_xyz123.zam456`. Somit kann Überschneidungen vorgebeugt werden, falls sich zwei Benutzer gleichzeitig an einem Linux-Rechner anmelden wollen.

  Die Kopier-Kommandos lauten dann - wenn eine Diskette als Schlüsselspeicher benutzt wird - für das Beispiel:

  ```
  scp userid@hostname:/media/floppy/userdat.pem.b64 \
      userdat.pem.b64_xyz123_zam456
  ```
  und
  ```
  scp userid@hostname:/media/floppy/passw.b64.pem.b64 \
      passw.b64.pem.b64_xyz123_zam456
  ```
- Der User-Record 'userdat*', der mit dem öffentlichem Schlüssel des Dispatch verschlüsselt worden war und dann in das Base64-Format konvertiert wurde, wird entschlüsselt:
  1. Dekodieren des Base64-Formates

```
$command = "openssl base64 -d -in $MAIN_DIR/$info_file_b64 \
           -out $MAIN_DIR/$info_file |";
open (REC,$command);
@ret= <REC>;
close (REC);
```

2. Entschlüsseln der Datei `userdat.pem_$userid_$host` mit dem Privaten Schlüssel des Dispatch, die Ausgabe erfolgt zeilenweise in das Array `@ret`

```
$decode_priv = "$MAIN_DIR/$PROGS/decrypt_priv";
$command = "$decode_priv $priv_adm $MAIN_DIR/$info_file |";
open (REC,$command);
@ret= <REC>;
close (REC);
```

- Aus diesem Array wird der Benutzername herausgefiltert und somit das spezifische Benutzer-Verzeichnis gebildet

```
# Name des Benutzers auslesen aus verschluesseltem User-Record (1.Zeile),
# z.B. T.Mustermann
# -> Bildung des User-Directories

$line = $ret[0];
chop($line);
$line =~ /^(\w+)\s*\|\s*(.*)$/;
($key, $value)  = ($1,$2);
$value =~ /([A-Z]).* (.*)/;
$name = "$1.$2";
$USER = "$DIR_USER/$name";
```

- Für die weiteren Kontrollen ist der User-Record relevant, der im Benutzer-Verzeichnis des Dispatch hinterlegt ist

1. Existiert ein Unterverzeichnis `$HOME/Users/$name`?
   (hier: `$HOME/Users/T.Mustermann`)
   Wenn nicht, wird der Login-Vorgang abgebrochen, da der Benutzer nicht bekannt ist und sich demnach auch nicht einloggen darf.
2. Wenn das Verzeichnis existiert, wird der User-Record gelesen.

```
# User-Record (Datei 'user_daten') des Benutzers lesen,
# der in der Benutzerverwaltung hinterlegt ist:
# -> Hash 'user_rec'

open(INFH, "<$USER/user_daten");
@user_dat = <INFH>;
close (INFH);

foreach $line (@user_dat) {
    chop($line);
    $line =~ /^(\w+)\s*\|\s*(.*)$/;
    ($key, $value)  = ($1,$2);
    $user_rec{$key} = $value;
}
```

3. Es wird geprüft, ob die von dem Benutzer über die Tastatur eingegebene UserID in der Zeile der nutzbaren UserIDs ("`UserID | ...`") vorhanden ist.

```
# Alle erlaubten UserIDs
@userids = split(" ",$user_rec{Userid});

# Vergleich mit der eingegebenen UserID
$id_flag = 0;
foreach $id (@userids) {
    if ($id eq $userid) {
```

```
            $id_flag = 1;  # User darf sich mit dieser UserID anmelden
            last;
        }
    }
    if (not $id_flag) {
        print "Returnwert:\nFailed ID";
        exit;
    }
```

4. Gibt es den Rechner an dem sich der Benutzer einloggen möchte im User-Record? (Zeile "`Host | ...`")

```
# Alle erlaubten Host
@hosts = split(" ", $user_rec{Host});
$h_flag = 0;
foreach $h (@hosts) {
    if ($h eq $host) {
        $h_flag = 1;
        last;
    }
}
if (not $h_flag) {
    print "Returnwert:\nFailed Host";
    exit;
}
```

5. Sind die Authentisierungsdaten noch gültig?
   (Eintrag in der Zeile "`gueltig_bis | ...`")

⇒ Trifft eine der Kontrollen nicht zu, wird das Programm sofort mit dem entsprechenden Fehlercode beendet.
An dieser Stelle besteht die Möglichkeit, weitere Kontrollen - besonders auch zur Autorisierung - einzubauen.

- Passwort vom USB-Stick/ von der Diskette entschlüsseln und mit dem eingegebenen vergleichen
  1. Die Datei wird aus dem Base64-Format zurück konvertiert und dann mit dem Privaten Schlüssel des Dispatch entschlüsselt.
  2. Diese temporäre Datei wird wieder base64-dekodiert und dann mit dem öffentlichen Schlüssel des Benutzers aus dem entsprechenden User-Record entschlüsselt. Läßt sich die Passwortdatei nicht entschlüsseln, dann wurde das Passwort nicht mit dem originalen privaten Schlüssel des Benutzers verschlüsselt. Der Login-Vorgang wird mit einem entsprechenden Fehlercode abgebrochen.
  3. Das entschlüsselte Passwort wird mit dem eingegebenen Passwort verglichen. Das Programm endet entsprechend mit `SUCCESS` oder `FAILED`.

## 8.5 Authentisierungsserver

Die eigentliche Authentisierung findet auf dem Server des Dispatch-Systems statt. Zu diesem Zweck ist dort eine spezielle UserID "`slnl`" eingerichtet, die Zugriff auf die Benutzerdaten des Dispatch hat. Diese UserID dient ausschließlich der Authentisierung der Benutzer für einen Login innerhalb der Workstationgruppe.
Bei der Ausführung des PAM-Moduls `login` wird über ssh (siehe 8.4.1) die Login-Anfrage an den User `slnl` beim Dispatch übermittelt. Damit dies ohne Interaktion möglich ist, muss für jeden Client der Workstation-Gruppe unter `slnl` ein Eintrag in der Datei `.shosts` vorhanden sein:

Inhalt der Datei `.shosts`

```
wsg-client1   root
```

```
wsg-client2   root
 ...
wsg-clientn   root
```

Die SSH Host Keys der Workstation-Gruppen-Clients (die öffentlichen SSH-Schlüssel von root) müssen beim Dispatch hinterlegt sein (in der Datei `/etc/ssh/ssh_known_hosts`). Außerdem sind in der ssh-Konfiguration beim Dispatch folgende Angaben erforderlich:

Inhalt der Datei `/etc/ssh/sshd_config`

```
 ...
HostBasedAuthentification   yes
RhostsRSAAuthentification   yes
 ...
```

So wird sichergestellt, dass von jedem der in der Datei `.shosts` aufgeführten Rechner ein durch den Superuser `root` gesendetes ssh-Kommando unter `slnl` akzeptiert wird.
Die Kommunikation läuft dabei auf folgende Weise ab: Der Workstation-Gruppen-Client fordert den sshd-Daemon auf der Dispatch-Seite zum Senden einer Zufallszahl auf, die dann mit dem privaten Schlüssel des Superusers `root` auf dem Client-Rechner signiert wird. Diese Zufallszahl kann mit dem zugehörigen öffentlichen Schlüssel des Clients wieder entschlüsselt werden und somit ist der sendende Client-Rechner authentisiert. Für die weitere Kommunikation zwischen den beiden Rechnern wird ein sogenannter Session-Key ausgehandelt, über den der Datentransfer verschlüsselt abgewickelt wird.

Damit mit dem ssh-Befehl keine Kommandos abgesesetzt werden können, die nicht vorgesehen sind, ist die Login-Shell des Benutzers durch eine Restricted Shell (Datei `restrict.sh`) ersetzt worden. Diese akzeptiert nur den Befehl `check_login.pl`.
Die entsprechenden Log-Informationen werden unter `/var/log/slnl` geschrieben. Das Skript `restrict.sh` wird unter `/etc/security/bin` abgelegt.

Die RSA-Authentisierung des Workstation-Gruppen-Clients zusammen mit der Restricted Login-Shell erscheinen für den Zugriff auf den Dispatch-Server als ausreichend. Durch die Nutzung des SSH-Daemons wird außerdem der Einsatz eines zusätzlichen Serverdienstes auf dem Dispatch-Rechner vermieden. Ein solcher Dienst könnte zum Beispiel mit Hilfe des Perl-Moduls `IO::Socket` (Objekte `IO::Socket::INET`) bereit gestellt werden. [Lit.16]

# Kapitel 9

# Tests

Das Testen der erstellten Programme und Funktionen erfolgte in der Entwicklungsphase nach der Bottom-Up-Methode. Dabei wurden zunächst die einzelnen Bausteine jeweils separat getestet und erst später im Zusammenspiel. Dieses Vorgehen bot sich an, da viele verschiedene Aufgaben an unterschiedlichen Stellen im System zu erfüllen waren (Verschlüsseln und Entschlüsseln, das PAM-Modul, das Kontrollprogramm), bei denen Fehler auftreten konnten. Nach den erfolgreichen Einzeltests wurden die Programme über die Client-Server-Kommunikation in realitätsnaher Umgebung getestet.

Der Test der Anwendung des Logins mit erweiterter Benutzer-Authentisierung erfolgte in einer kleinen Workstation-Gruppe mit drei Client-Rechnern und dem Dispatch-Server (mit der UserID `slnl`).

**Funktionstest**

Die Eingabe einer gültigen UserID und des richtigen Passwortes wurde jeweils mit der Verwendung eines USB-Sticks und mit einer Diskette für den Fenster- und den Kommandozeilen-Login getestet. Ergebnis: Der Benutzer wird zum System zugelassen.

Ebenfalls getestet wurde der Fall, dass zwei Benutzer sich gleichzeitig mit derselben UserID einloggen. Ergebnis: Auch dieser Fall wird korrekt behandelt.

**Fehlertests**

Fehlerhafte Benutzereingaben:

- Eingabe einer UserID, die nicht in dem User-Record steht
- Login-Versuch von einem Rechner aus, der nicht im User-Record steht
- Falsches oder kein Passwort eingegeben

In diesen Fällen wird der Login-Vorgang abgebrochen. Es erfolgt ein Eintrag in der entsprechenden Log-Datei `/var/log/slnl/check_login_*`. Um es Angreifern zu erschweren, die richtige UserID-Passwort-Kombination herauszufinden wird der genaue Grund für den missglückten Login-Versuch nicht in einer präzisen Fehlermeldung mitgeteilt.

Weitere Fehlerfälle:

- Keinen USB-Stick bzw. Diskette benutzt
- Dateien sind weder auf dem USB-Stick, noch auf der Diskette
- Dateien lassen sich nicht entschlüsseln

In diesen Fällen wird der Login-Vorgang abgebrochen und es wird die entsprechende Fehlermeldung zur Information des Benutzers ausgegeben.

Zusammenfassend lässt sich sagen, dass mit den Tests bewiesen wurde, dass das entwickelte Konzept realisierbar und funktionsfähig ist.

# Kapitel 10

# Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Konzept vorgestellt, wie man die Sicherheit für den Login in den Workstation-Gruppen des Forschungszentrum Jülich mit einer erweiterten Benutzer-Authentisierung erhöhen kann. Dazu wurde die bestehende Benutzerverwaltung und das auf Linux-PAM beruhende Login-Verfahren beschrieben. Um die verwendeten Verschlüsselungsmechanismen zu verstehen, wurde eine Einführung in die Kryptographie und in das bekannteste und auch implementierte RSA-Verfahren gegeben. Für die Speicherung der RSA-Schlüssel und des Referenzpasswortes wurden traditionelle und moderne Datenträger diskutiert.
Die Implementierung erfolgte für den Login auf Fenster- und Kommandozeilenebene und umfasste das Entwickeln eines eigenen PAM-Moduls, kryptographischer Programme und einer vorerst einfachen Struktur für die Benutzerverwaltung.

Das Verfahren ist sicherer, da für den Login an einem Linux-Rechner zusätzlich der Datenträger im Besitz des Benutzers sein muss und er das dort gespeicherte und vor missbräuchlichem Lesen geschützte Passwort wissen muss.
Das Verfahren wurde mit einfachen Mitteln realisiert, da als Schlüsselspeicher gängige Datenträger verwendet werden und nicht in zusätzliche Hardware - wie bei der Verwendung von Smartcards - investiert werden muss.

Nachteilig ist, dass die Daten von dem persönlichen Schlüsselspeicher, wozu das Zugangspasswort und der private Schlüssel des Benutzers zählen, abgefangen oder mitgehört werden können, und nicht fest auf dem Speicher abgelegt sind, wie es bei den Smartcards möglich ist. Aus diesem Grund werden diese Daten verschlüsselt und sind nur vom Dispatch entschlüsselbar. Somit bietet dieses Verfahren für den Login an den Linux-Rechnern der Benutzer ausreichende Sicherheit.

Da in der Workstation-Gruppe von allen Clients aus auf das selbe Home-Verzeichnis eines Nutzers zugegriffen wird, kann sich ein Benutzer mittels `ssh` auf allen Clients einloggen.
Das implementierte Verfahren stellt ein sogenanntes `Single Sign Logon System` dar. Dabei erfolgt die Identifikation des Benutzers nur zum Zeitpunkt des ersten Zugriffs auf einen Rechner der Gruppe und er muss sich nicht erneut authentisieren, wenn er an einem anderen Rechner in der Workstation-Gruppe arbeiten möchte.

Für das Arbeiten in Workstation-Gruppen ist eine Netzwerkanbindung unverzichtbar, da die Home-Filesysteme der Benutzer vom Gruppenserver zur Verfügung gestellt werden. Unter der Vewendung des NIS-Protokolls (bisheriges Vorgehen) wird die Benutzerverwaltung ebenfalls über den Gruppenserver abgewickelt.
Für das implementierte Verfahren wird die Benutzerverwaltung vom Dispatch-Server ergänzt, der die Login-Anfragen annimmt und kontrolliert, ob die Login-Daten korrekt sind.

Dieser Dispatch-Server verwaltet die Benutzerdaten in einer Directory-Struktur. Denkbar wäre an dieser Stelle auch die Verwendung von LDAP[1], einem Directory-Zugangsprotokoll, das ebenfalls auf Client-Server-Basis arbeitet und das zu einem Standard bezüglich der Verwaltung von Benutzerdaten geworden ist. Damit könnte auf die Benutzung des NIS verzichtet werden.
In jedem Fall sollte ein Backup-Server bereitstehen, der es im Problemfall den Benutzern ermöglicht, sich auf den Gruppen-Clients anzumelden.

In Zukunft sollte noch die Änderung des Passwortes unter der Verwendung des persönlichen Datenträgers ermöglicht werden.

[1] Lightweight Directory Access Protocol

# Literaturverzeichnis


[Lit. 1] SuSE Linux Enterprise Server für Unix-Administratoren
(Lernunterlage - Artikelnr. 45451-1, 2002)

[Lit. 2] http://www.kernel.org/pub/linux/libs/pam

[Lit. 3] http://www.kernel.org/pub/linux/libs/pam/pre/doc/rfc86.0.txt.gz

[Lit. 4] http://www.mathematik.de/spudema/spudema_beitraege/beitraege/
hillebrand/mathe2002/publickey.htm
http://www.regenechsen.de/krypto/rsa.php

[Lit. 5] Beutelspacher, Schwenk, Wolfenstetter
Moderne Verfahren der Kryptographie
vieweg, 1999, 3. Auflage

[Lit. 6] Arno Lindhorst
Sichere E-Mails mit PGP
verlag moderne industrie Buch AG & Co. KG, 2002, 1. Auflage

[Lit. 7] Martin Sägesser, Mario Strasser
Linux Netzwerk-Login mit RSA-Smartcards, 2001
http://home.zhwin.ch/ sna/DA/Sna1_2001.pdf

[Lit. 8] OpenSSL-Handbuch
http://www.dfn-pca.de/certify/ssl/handbuch/

[Lit. 9] http://www.openssh.org/manual.html

[Lit.10] H.R.Hansen, G.Neumann
Wirtschaftsinformatik I
Lucius & Lucius, 2001, 8. Auflage

[Lit.11] http://www.linux-magazin.de/Artikel/ausgabe/2002/05/smartcard/smart.html

[Lit.12] Andrew G. Morgan
The Linux-PAM Module Writers' Guide, 2002
http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam_modules.html

[Lit.13] http://www.hmug.org/man/3/RSA_generate_key.html

[Lit.14] http://www.hmug.org/man/3/RSA_private_encrypt.html

[Lit.15] http://www.hmug.org/man/3/RSA_public_encrypt.html

[Lit.16] http://search.cpan.org/ nwclark/perl-5.8.5/ext/IO/lib/IO/Socket/INET.pm